Herramienta para la abstracción de complejidad de configuración de switches de acceso

Jordi Mèlich Pelegrí

Director: Enric Guitart Baraut

Universitat de Lleida
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Trabajo de final de Grado

2017

Índice general

Índice de figuras

Capítulo 1

Introducción

Este proyecto surge del convenio de colaboración firmado entre la Universitat de Lleida y Alcatel-Lucent en el ámbito de Redes y Comunicaciones. El proyecto se enmarca dentro del desarrollo de soluciones SDN (*Software Defined Networks*), uno de los principales objetivos del convenio de colaboración, y para su desarrollo se empleará la dotación de equipos Alcatel-Lucent OS6860.

1.1. Objetivos

El objetivo del trabajo consiste en diseñar e implementar una herramienta de configuración básica de *switch* de acceso con el objetivo de abstraer complejidad de configuración de equipos de red y permitir que las tareas básicas de configuración de los puertos pueda realizarse por personal no especialista en redes y comunicaciones.

Para ello se desarrollará una plataforma web capaz de ofrecer al usuario un menú de configuración sencillo para cada punto de red y aplicar los cambios automáticamente en el equipamiento. Por un lado se abstraerá complejidad de configuración evitando la utilización del terminal de los equipos. Por otro lado, se evitará aplicar cambios en la configuración a nivel de puerto del equipo ya que también se manejará el patrimonio arquitectónico y equipamiento de red de la entidad, por lo tanto, se podrá vincular un punto de red a un puerto en concreto del equipamiento.

En relación al patrimonio arquitectónico de la entidad, se gestionará: los campus, edificios, plantas, salas de equipamiento y puntos de red. En cuanto a equipamiento tecnológico se gestionará: el equipamiento, las tarjetas de estos últimos y los puertos. Así pues, estableciendo una relación entre cada punto de red y cada puerto del equipamiento se podrá obtener el equipo en concreto al que se le aplicará los cambios en la configuración.

En base a los objetivos descritos se identifican tres bloques principales de trabajo:

- ***Back-end***: Parte del software la cual será consumida por el *front-end*, además se encargará de garantizar la persistencia y de la comunicación con el equipamiento con el objetivo de hacer efectivos los cambios de configuración.
    - **Base de datos:** Software que garantizará la persistencia y las relaciones entre las entidades listadas en el parágrafo anterior y será manejada por el *back-end*.
- ***Front-end***: Parte del software con la que el usuario interactuará. Se encargará de sintetizar y traducir los datos entre usuario y *back-end* en ambas direcciones.
- **Comunicación con el equipamiento**: Establecer la comunicación entre el *back-end* y los dispositivos Alcatel OS6860 para explotar la funcionalidad SDN(*Software Defined Network*) que ofrecen.

Cabe añadir que para la implementación del proyecto, tanto la base de datos, como el *back-end* y el *front-end* serán con licencia libre.

1.2. Estructura

El desarrollo del proyecto se dividirá en etapas coincidiendo con las propuestas por la metodología de desarrollo clásica o en cascada [23, 22].

- **Análisis**: Esta tarea constituye la fase inicial en el desarrollo del proyecto en la cual se analizarán las distintas tecnologías de las que disponemos actualmente y decidir, atendiendo a sus pros y contras, cuál será la que mejor se adapte y resuelva cada uno de los requerimientos que exige el proyecto en cuestión.

- **Diseño**: En la tarea de diseño se realizarán los esquemas y diagramas requeridos para la siguiente fase, la de implementación, en base a los resultados obtenidos en el estudio sobre las distintas tecnologías en la fase de análisis. De este modo, se optará por las que mayor ventaja y utilidad aporten para la realización del proyecto.
- **Implementación**: En base a las tecnologías resultantes de la fase de análisis y los esquemas realizados en la fase anterior, se implementarán las ya mencionadas partes esenciales que corresponden al *back-end*, *front-end* y la comunicación con el equipamiento que constituirán la solución final del proyecto.
- **Implantación**: En la fase de implantación se desplegará la solución software en el equipo que la albergará y devendrá funcional para el usuario final.

1.3. Planificación temporal de tareas

En cuanto a la planificación temporal de las tareas se ha recurrido a un diagrama de Gantt para representarlas desglosadamente:

1.3.1. Fase de Análisis

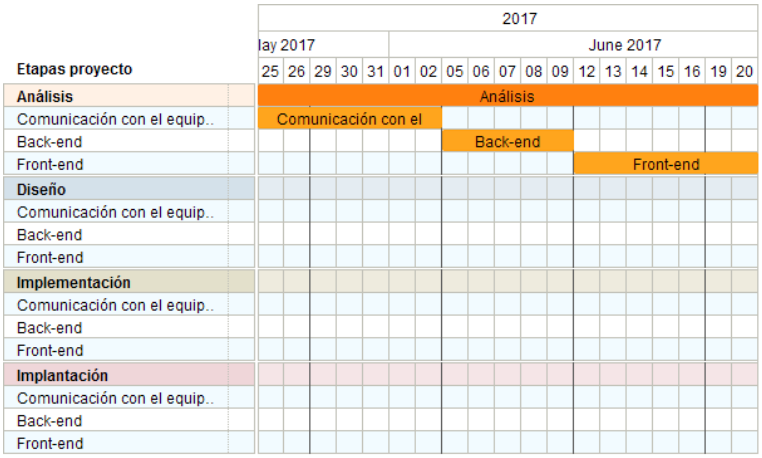


Figura 1.1: Diagrama de Gantt: Fase de Análisis

1.3.2. Fase de Diseño

Figura 1.2: Diagrama de Gantt: Fase de Diseño

1.3.3. Fase de Implementación

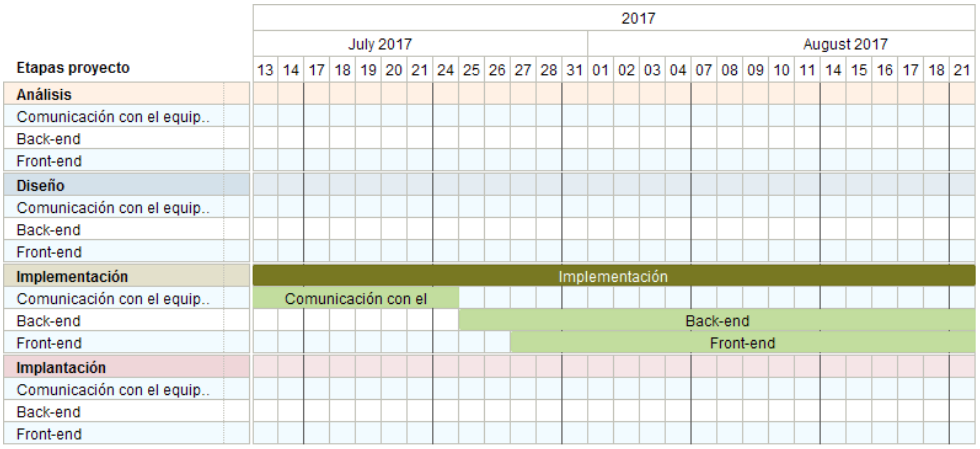


Figura 1.3: Diagrama de Gantt: Fase de Implementación

1.3.4. Fase de Implantación

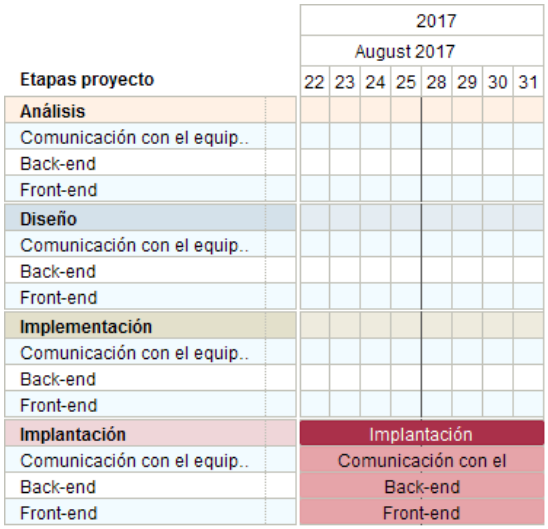


Figura 1.4: Diagrama de Gantt: Fase de Implantación

Capítulo 2

Análisis

Esta tarea constituye la fase inicial en el desarrollo del proyecto la cual consiste en analizar las distintas tecnologías de las que disponemos actualmente para la implementación de cada uno de los componentes y decidir, atendiendo a sus pros y contras, cuál es la que mejor se adapta y resuelve cada uno de los requerimientos que exige el proyecto en cuestión.

2.1. Comunicación con el equipamiento

En este apartado se analizarán los mecanismos disponibles a través de los cuales se puede interactuar con los equipos para manipular los parámetros de configuración.

Es importante mencionar que los equipos que se utilizarán para este proyecto (Alcatel OS6860) disponen de distintas tecnologías de comunicación para explotar su configuración y monitorización de forma remota. Estas tecnologías son SNMP(*Simple Network Management Protocol*) y una API REST(*REpresentational State Transfer*), ambas ofrecidas por sus *Web Services* para explotar la funcionalidad SDN(*Software Defined Network*).

2.1.1. SNMP

El protocolo SNMP fue introducido en 1988 y consiste en un estándar de gestión para equipos de red. El objetivo de SNMP es proveer a los administradores de dichos dispositivos un conjunto de operaciones para que estos puedan ser configurados y monitorizados de manera remota. Por ejemplo, se puede utilizar para conocer la velocidad de transferencia

de un puerto en concreto, apagarlo e incluso para consultar la temperatura de un equipo y programar un aviso en caso de que se exceda un límite previmaente establecido. Cabe añadir que este protocolo se suele vincular a *routers* y *switches* aunque puede ser utilizado por ordenadores personales, servidores e incluso impresoras; en resumen, por cualquier dispositivo cuyo sistema operativo permita la integración de SNMP como se refleja en la Figura 2.1.

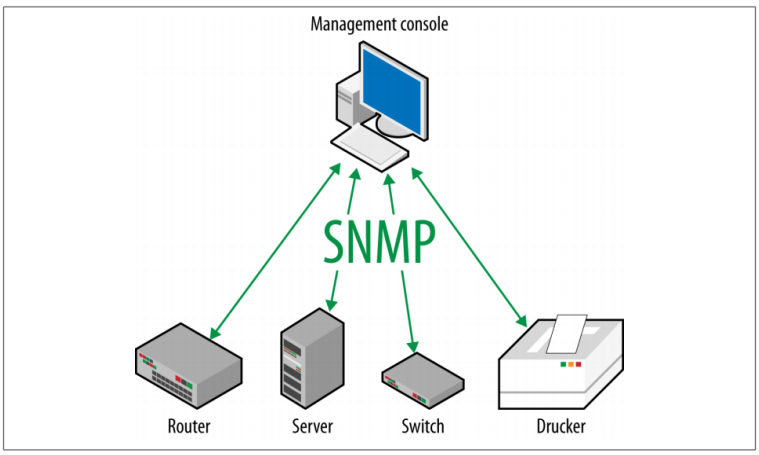


Figura 2.1: Ejemplos de utilización de SNMP

En la Figura 2.2 se muestra la arquitectura SNMP cuyos componentes se describirán en los siguientes apartados.

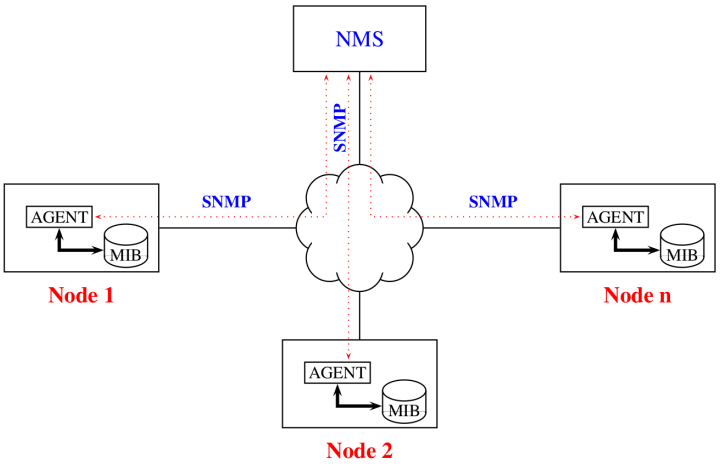


Figura 2.2: Ejemplo de la arquitectura de componentes de SNMP

NMS

El NMS(*Network Management Station*) es un dispositivo de propósito general que ejecuta el software de gestión. Se encarga de interactuar con el agente de cada uno de los nodos ya sea para consultar o asignar un valor, tal como se puede observar en la Figura 2.3 y Figura 2.4 respectivamente. También se encarga de recibir las *traps* generadas por el Agente, que consisten en un tipo de notificación que se lanza una vez superado un umbral definido para el nodo y/o al detectar cambios en algún parámetro.

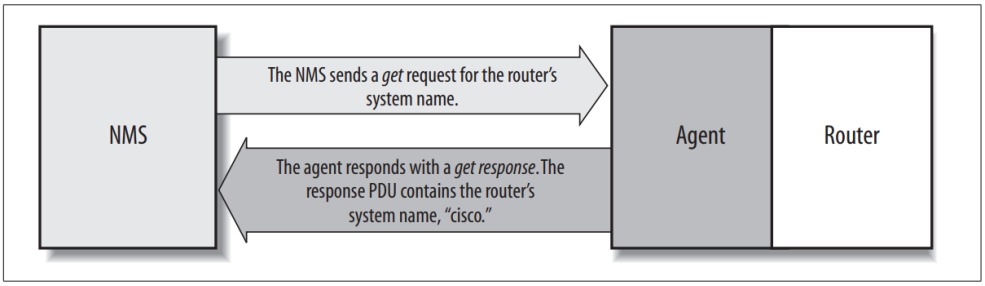


Figura 2.3: Diagrama de consulta de valores NMS-Agente

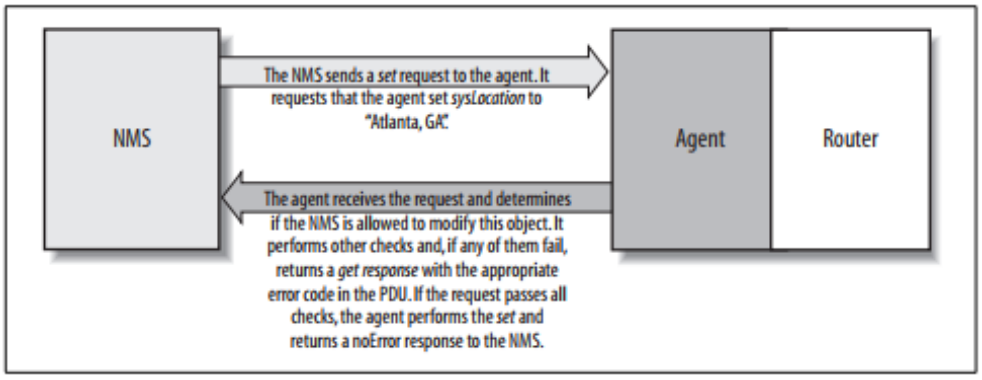


Figura 2.4: Diagrama de asignación de valores NMS-Agente

Agente

El agente es el servicio que ejecuta cada uno de los nodos para ser gestionado por el NMS. Se encarga de gestionar las peticiones generadas

por el NMS además de generar *traps* y notificar al NMS en caso de que haya cambios en la configuración. Como se puede apreciar en las Figuras 2.3 y 2.4, el Agente actúa cómo intermediario entre el NMS y el Router.

MIBs

Las MIBs(*Management Information Base*) corresponden a las variables u objetos que son manipulados, ya sea para consultar su valor o para editarlo. Estas variables representan los parámetros de la configuración y el estado de un equipo de red.

La estructura que forma cada una de las MIBs viene definida por el formato SMI(*Structure of Management Information*) y, por lo tanto, todas disponen de los siguientes atributos:

- **Nombre** También llamado OID(*Object Identifier*) identifica cada una de las MIBs como un objeto único disponible. Un ejemplo de ello se refleja en la Figura 2.5 y se representa en ambas de las dos siguientes formas:
  - **Formato numérico:** Corresponde a la concatenación del OID del nodo padre recursivamente hasta el nodo raíz y el OID que se le haya asignado a la MIB en cuestión, separados por el carácter ".".
  - **Cadena de caracteres:** Corresponde al nombre formando una cadena de caracteres que resulta "legible para los humanos".
- **Tipo y Sintaxis:** El tipo de datos que representa cada objeto viene dado por un subtipo de ASN.1(*Abstract Syntax Notation One*). ASN.1 es una forma de especificar cómo los datos son representados en el NMS y los Agentes en el contexto de SNMP. La ventaja de utilizar este formato es que la notación resulta independiente de la plataforma donde se esté ejecutando. De este modo se garantiza que los valores de los datos son compatibles con los demás componentes que utilizan el protocolo.
- **Codificación:** El modo de codificación define de que manera cada uno de los objetos resulta codificado/decodificado en el momento de ser transmitido a través de cualquier medio como puede ser Ethernet.

El conjunto de MIBs se estructuran en forma de árbol como se puede apreciar en la Figura 2.5, aunque únicamente los nodos hoja representados en la Figura 2.6 corresponden a MIBs cuyos valores pueden ser manipulados, ya sea para obtener el valor o editarlo.

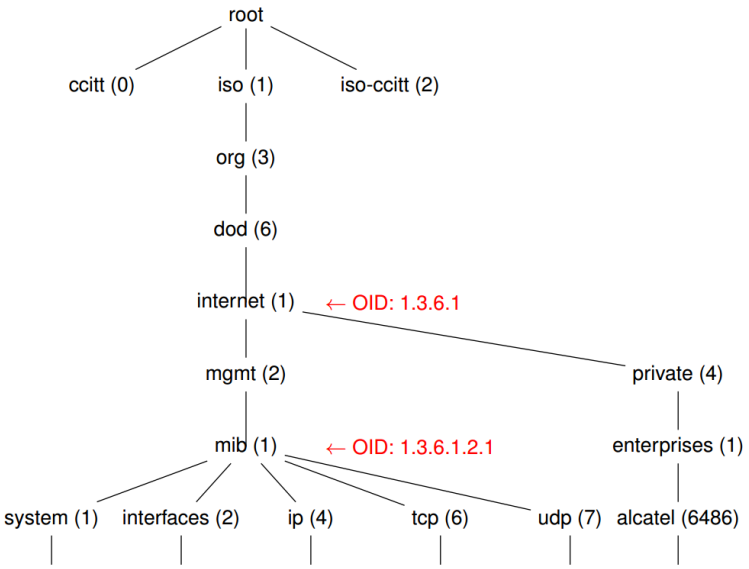


Figura 2.5: Árbol de MIBs

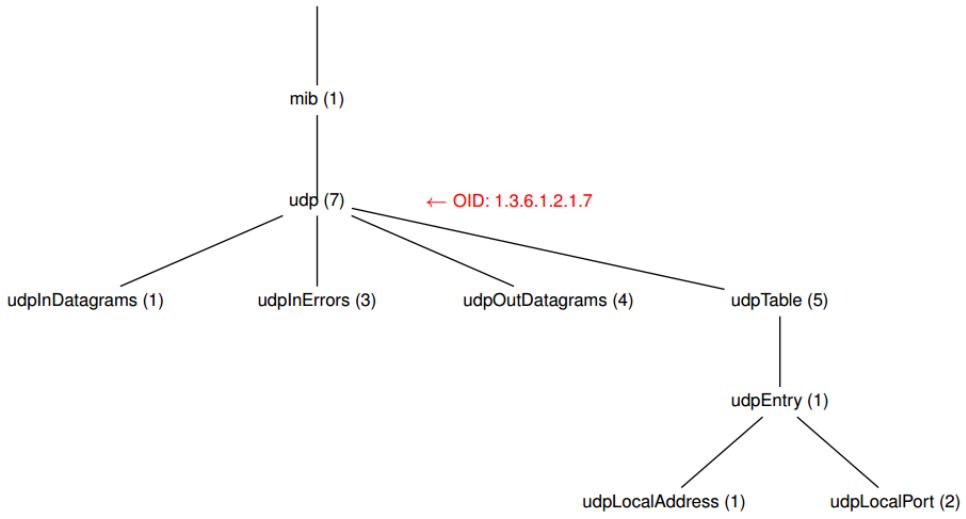


Figura 2.6: MIBs en nodos hoja

Protocolo

El protocolo define las reglas de comunicación entre los Agentes y el NMS. La Figura 2.7 refleja la interacción NMS-Agente dónde el NMS se encarga de realizar las peticiones al Agente y de recibir las *traps*, mientras que el Agente, por su parte, se encarga de resolver las peticiones y generar los *traps*.

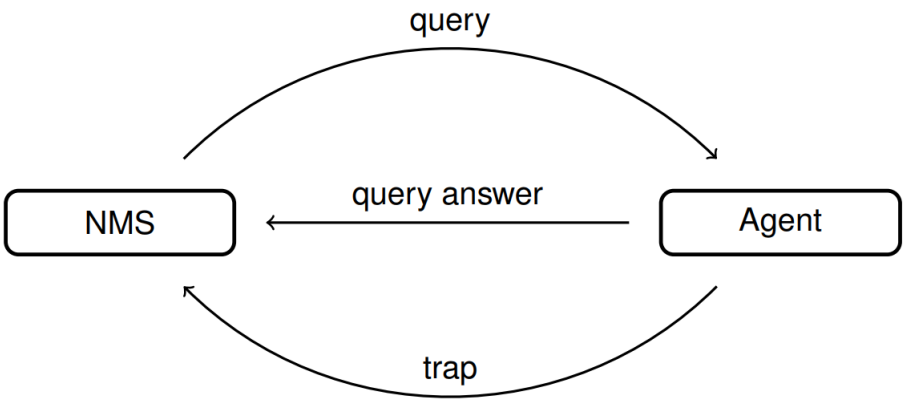


Figura 2.7: Diagrama genérico de la interacción NMS-Agente

Las comunicaciones del protocolo se realizan a través del protocolo UDP y el puerto 161, aunque las *traps* se realizan a través del puerto 162.

Además dispone de tres versiones de utilización, todas ellas soportadas por los equipos que se utilizarán para el proyecto. Las características de cada una de ellas son las siguientes:

- **SNMPv1:** Esta versión corresponde a la implementación original del protocolo. Se caracteriza por disponer de las operaciones o PDUs(*Protocol Data Unit*): Get, Set, GetNext y Trap sobre las MIBs. Proporciona un sistema rudimentario de seguridad mediante la definición de *communities*. A cada una de ellas se le permite un conjunto de PDUs. De esta manera se gestionan los permisos de lectura/escritura o traps para cada NMS el cual tiene una *community* asociada.

- **SNMPv2:** La segunda versión resulta compatible con la anterior en cuanto a las PDU y la seguridad basada en *communities*. Además aporta mejoras en cuanto al manejo de la información como por

ejemplo la adición de nuevas PDU, obviamente incompatibles con la version anterior.

- **SNMPv3:** La tercera versión se caracteriza por el acceso basado en usuario en lugar de *communities*, además de permitir restricciones a nivel de tuplas para las MIBs representadas en forma de tablas. También aporta las siguientes nuevas funcionalidades respecto a las versiones anteriores:
  - Comprueba la integridad del mensaje.
  - Permite especificar un intervalo de tiempo para las peticiones a nivel de PDU. Si se realiza una petición fuera de la marca de tiempo, esta será ignorada.
  - Permite el cifrado de paquetes de transmisión.
  - Permite la autenticación de usuarios.

2.1.2. SDN

Este corresponde al segundo mecanismo de configuración remota que ofrecen los equipos. Como se detalla en [19], SDN consiste en un paradigma cuyo objetivo es separar la integración vertical de la arquitectura de los equipos. De esta manera se segregan las capas de control y la de datos e interactuando con la primera de ellas, la red deviene programable.

Arquitectura y Componentes

Existen tres capas bien diferenciables, detalladas a continuación y reflejadas en la Figura 2.8:

- **Capa de Aplicación:** Esta capa cubre una serie de aplicaciones centradas en los servicios de red y son principalmente aplicaciones de software que se comunican con la capa de control. En resumen, las aplicaciones de gestión de red que interactuan con los equipos. En esta es donde se ubica la aplicación resultante del proyecto.
- **Capa de Control:** Representa el núcleo de SDN, esta capa consta de un controlador centralizado que atiende las peticiones de la capa de Aplicación y gestiona los dispositivos de red a través de protocolos estándar.

- **Capa de Datos:** Esta capa representa la infraestructura en sí e incluye cualquier dispositivo de red que pueda ser gestionado.

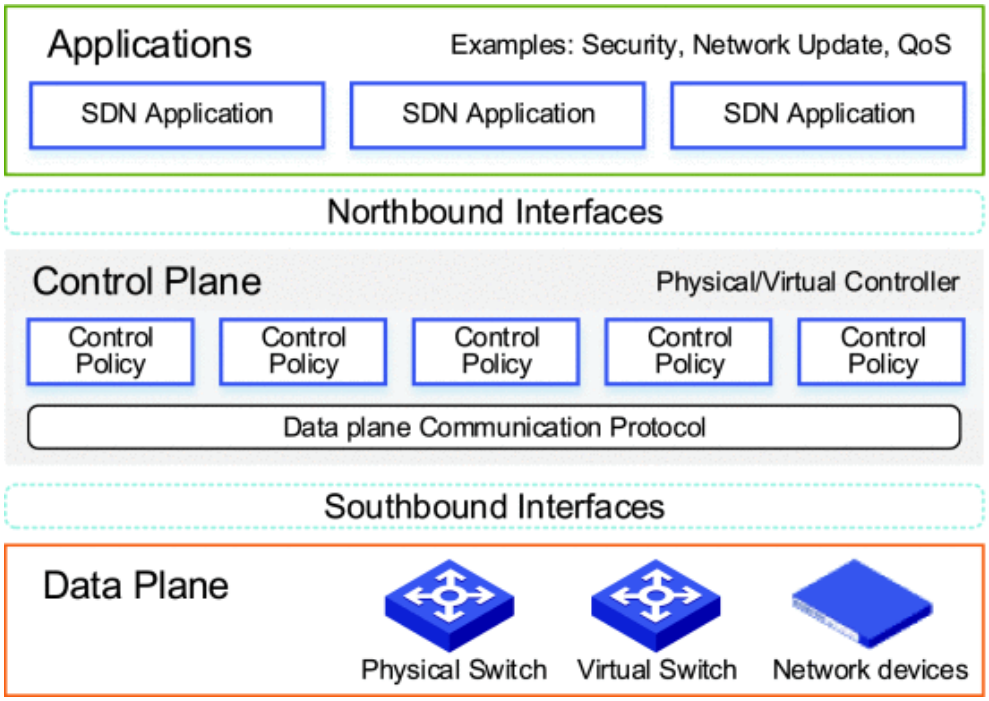


Figura 2.8: Arquitectura y componentes de SDN

En el contexto del proyecto la capa de Control es la que nos interesa analizar más a fondo, y es lo que se va a realizar en el siguiente apartado.

REST

Así pues, según la documentación oficial de los dispositivos[16], la capa de Control ofrece un servicio web REST accesible desde una API de Python proporcionada por Alcatel o mediante peticiones HTTP(*HyperText Transfer Protocol*). Estas peticiones se pueden realizar a través de HTTP o a través de su versión securizada HTTPS y permiten manejar la configuración del equipamiento ya sea manipulando directamente a las variables MIB utilizadas por el ya mencionado protocolo SNMP o mediante comandos de CLI(*Command Line Interface*). Cabe añadir que los equipos permiten un máximo de 4 sesiones simultáneas.

Las características que ofrece un servicio web REST són las siguientes:

- **Arquitectura Cliente-Servidor:** El servidor es quien provee los recursos, y estos son consumidos por uno o más clientes.
- **Sin Estado:** Cada una de las peticiones ha de resultar autosuficiente e independiente, de este modo se evita interferencias con las demás y posibles dependencias entre ellas.
- ***Cacheable:*** En caso de emitir una petición repetidamente o habitualmente, se almacena el resultado de esta para no volverla a generar y evitar cálculos redundantes.
- **Nombres de Recursos:** Se establece un nombre canónico para cada uno de los recursos mediante URIs(*Uniform Resource Identifier*) para definir su ubicación mediante una URL única.
- **Interfaz Uniforme:** Todos los recursos se consideran sujetos y se interactua con ellos mediante verbos HTTP(GET, POST, PUT, DELETE).
- **Tipos de Medios:** Permite diferenciar el tipo de recursos con los que se esté tratando.

La seguridad y autenticación se mantiene mediante sesiones en la parte del servidor y *cookies* por parte del cliente.

REST admite dos contextos de acceso a la configuración para los equipos utilizados en el proyecto:

1. **Configuración a través de MIB**

   La lectura y modificación de las MIBs resulta una de las posibilidades de configuración y gestión remota que ofrece el ya mencionado servicio web REST disponible en los equipos y, por lo tanto, constituye una opción a tener en cuenta para su gestión de manera remota. En este caso, desde la aplicación que se desarrollará para el proyecto.

   Para acceder a las MIB mediante el servicio web REST se utilizarán peticiones HTTP parametrizadas con un método(GET, PUT, POST, DELETE), la dirección del equipo a monitorizar seguido

del contexto de configuración(mib, info, cli) y la variable MIB a acceder; y en caso que se desee modificar la configuración, los nuevos valores.

En el siguiente ejemplo se muestra una petición HTTP para crear una nueva VLAN2 con descripción: VLAN-2 en el equipo que responde a la dirección IP 192.168.1.1 utilizando MIBs:

```
PUT https://192.168.1.1/mib/vlanTable?
mibObject0=vlanNumber:2\&
mibObject1=vlanDescription:VLAN-2
```

2. **Configuración a través de CLI**

   La alternativa es la utilización de CLI, que consiste en una interfaz proporcionada por los equipos accesible a través de la consola, Telnet y SSH para monitorizar y gestionar el sistema y su configuración.

   Añadiendo sentencias CLI a las peticiones realizadas contra el servicio web REST el sistema es capaz de sintetizar dicha petición, extraer la sentencia CLI y ejecutarla. Como respuesta se devolverá la salida por pantalla que genera la ejecución de esta en el terminal del equipo.

   Para ejecutar las sentencias CLI, se generará una petición HTTP con el método más adecuado según la finalidad de esta además de la dirección IP del dispositivo y el contexto, concatenando el comando CLI a ejecutar y substituyendo los espacios de este por el símbolo "+".

   El siguiente ejemplo muestra un caso de petición HTTP para obtener la salida por pantalla del comando de CLI `show vlan 5` el cual ejecutado directamente en el terminal del equipo mostraría los parámetros de configuración para la VLAN5:

   ```
   GET https://192.168.1.1/cli/aos?&cmd=show+vlan+5
   ```

2.2. *Back-end*

El *back-end* representa el programa ejecutado por el servidor y su cometido es atender a las peticiones realizadas por el cliente. Así pues, en esta fase de análisis se estudiarán las plataformas disponibles para el desarrollo de dicho *back-end*, exponiendo, para cada una, sus rasgos y características técnicas. De esta manera, en la siguiente fase de Diseño, se decidirá cuál es el que aportará una mayor ventaja en la implementación del proyecto.

El desarrollo del *back-end* normalmente se realiza bajo un *framework*. Dicho *framework* se escoge en base a la naturaleza de la aplicación a desarrollar, ya que este proporciona herramientas y funciones genéricas que puedan resultar útiles para distintos proyectos ademas de proporcionar una arquitectura estándar a la aplicación.

Así pues, se analizarán los *frameworks* realizados bajo los lenguajes de programación .NET, Java, Python, PHP y NodeJS ya que son los más utilizados hoy en día para el contexto de los *back-ends*. No obstante, se requerirá que ambos dispongan de soporte y mantenimiento actualmente.

A continuación se incluye una breve aproximación sobre cada uno de los lenguajes de programación en los que se basan los *frameworks* que serán analizados a posteriori:

Java

Java[21] es un lenguaje de programación diseñado en 1990 por Sun Microsystems a partir de C++. Entre 2006 y 2007, dicha empresa liberó partes de código cómo programario libre y actualmente es uno de los más utilizados debido a su flexibilidad y la capacidad de ejecutarse en *hardware* con arquitecturas distintas (multiplataforma). Además el código fuente escrito en Java guarda cierto parecido con otros lenguajes cómo por ejemplo C y C++.

Python

Es un lenguaje de programación de muy alto nivel, publicado en 1991 por la Python Software Foundation con licencia libre. Su filosofía de diseño consiste en tener una sintaxis legible y agradable, además de utilizar muchas menos líneas de código con el mismo fin que en otros lenguajes, incluso reduciendo el volumen total de estas a la mitad. Su sintaxis resulta parecida a otros lenguajes tales cómo C y Java. Por otra parte, Python[20] es un lenguaje interpretado, eso significa que su código fuente está traducido por el intérprete en el momento de su ejecución.

PHP

PHP[7](*Personal Home Page Tools*) es un lenguaje de propósito general y de código abierto con licencia libre que está especialmente orientado al desarrollo web. Fue publicado en 1995 por PHP Group. Su sintaxis recurre a C, Java y Perl. El objetivo principal de este lenguaje es permitir a los desarrolladores generar rápidamente aplicaciones web. Cabe añadir que es el lenguaje orientado a web más extendido por el momento.

NodeJS

NodeJS[18] fue publicado en 2009 por Ryan Lienhart Dahl bajo la licencia del MIT, y por ende licencia libre. NodeJS no es directamente un lenguaje de programación aunque su cometido es ejecutar código JavaScript (orientado a ser ejecutado por el cliente) en el lado del servidor.

Una vez presentados los principales lenguajes de programación en los que se basan los *frameworks* para *back-ends* nos centraremos en detallar los más extendidos y sus correspondientes características técnicas.

2.2.1. .NET

La plataforma .NET[5] fue presentada por Microsoft en el año 2000 y por aquel entonces fue el principal competidor de Java. Aunque sea software propietario y el proyecto se deba realizar con software libre, representa una de las alternativas en cuanto a los *back-ends* y merece tenerlo en cuenta.

.NET directamente ya es un *framework* y su arquitectura es bien sencilla, pues sólo consta de dos componentes básicos mientras que el resto resultan adicionales. El primero de ellos corresponde a CLR (*Common Language Runtime*) y representa el núcleo del entorno y es el encargado de controlar las aplicaciones en ejecución de manera que administra la memoria, la ejecución de subprocesos, la comunicación remota y la seguridad y solidez de las instrucciones en tiempo de ejecución. El segundo componente es la biblioteca de clases de .NET, que ofrece código ya probado y reutilizable pudiendo ser llamado desde las aplicaciones realizadas bajo este entorno.

En el esquema reflejado en la Figura 2.9 se representa la arquitectura básica de .NET así como alguno de sus componentes extra. Dejando de lado el sistema operativo (obviamente bajo Microsoft Windows) se puede apreciar cómo a partir de CLR y la biblioteca base se desprenden el resto de componentes del *framework*.

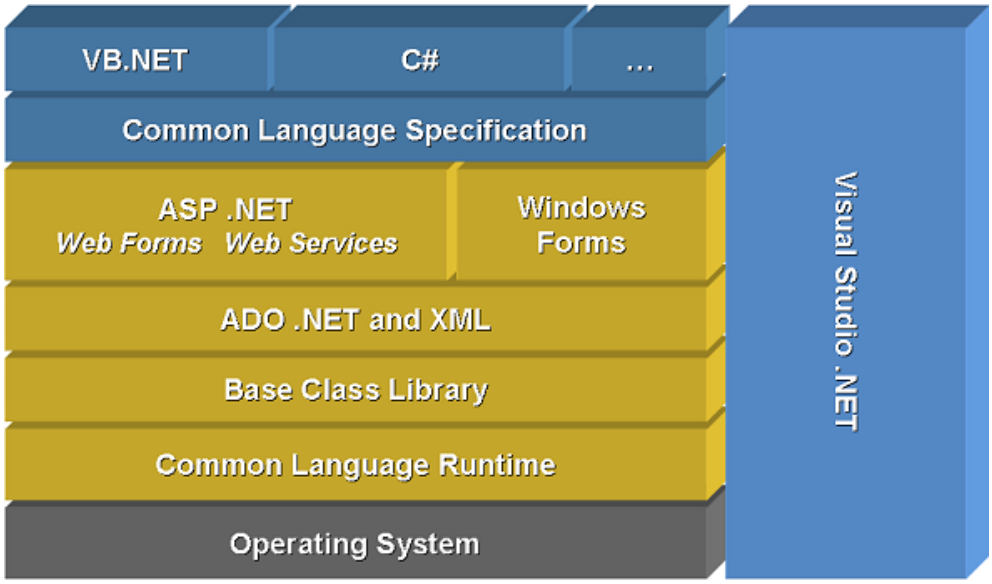


Figura 2.9: Arquitectura .NET

En el nivel superior de la biblioteca de clases, se encuentra ADO.NET (`ActiveX Data Objects`) y XML que constituyen el acceso a datos. Este acceso a datos tiene como objetivo establecer comunicación con una base de datos o bien con ficheros XML.

En el quinto nivel se encuentra ASP.NET y Windows Forms. Estos se encargan de sintetizar las instrucciones realizadas en CLS (*Common Language Specification*) para la realización de aplicaciones web o aplicaciones de escritorio respectivamente. ASP.NET es el que se explotaría en el contexto del proyecto debido a que está orientado a la programación web, ya sea en forma de servicios web o *Web Forms* (interfaz web).

El penúltimo nivel simboliza el traductor de sentencias en código fuente realizado por la capa superior, de manera que se puede realizar aplicaciones en distintos lenguajes de programación (deben de ser soportados por el *framework*) y estos serán traducidos al lenguaje natural del *framework*. Así pues, el *framework* ofrece las mismas funcionalidades para los distintos lenguajes de programación listadas a continuación:

- Gestión de memoria automática.

- Biblioteca de clases extensa. Esto supone una ventaja en tanto que el programador se ahorra el tener que definir clases genéricas para sus proyectos.
- Compatibilidad con versiones posteriores. De este modo el código fuente desarrollado será compatible con futuras versiones del *framework*.
- Multi-targeting. Consiste en que una misma aplicación pueda ser ejecutada en distintas plataformas de Microsoft (Windows 7, Windows 10, Xbox, Windows phone).

2.2.2. Spring MVC

Spring MVC[17](*Model-View-Controller*) es el *framework* más popular implementado bajo el *framework* Spring, y este, bajo el entorno Java. Las siglas MVC corresponden al tipo de arquitectura que este *framework* establece para los proyectos o aplicaciones que desarrolla.

Componentes y Arquitectura:

MVC es un patrón de arquitectura web, que define tres componentes principales para cada proyecto. Dichos componentes se describen a continuación y la arquitectura que forman se representa en la Figura 2.10:

- **Modelo:** El modelo representa el núcleo de la lógica de negocio y de los datos. Es el componente que define las propiedades y comportamiento de cada una de las entidades.
- **Vista:** La vista se encarga de transformar los datos del modelo en una representación visual en base a la petición generada por el usuario.
- **Controlador:** El controlador se encarga de manejar las peticiones recibidas y, en base a estas, coordina el Modelo y la Vista para ofrecer una respuesta coherente.

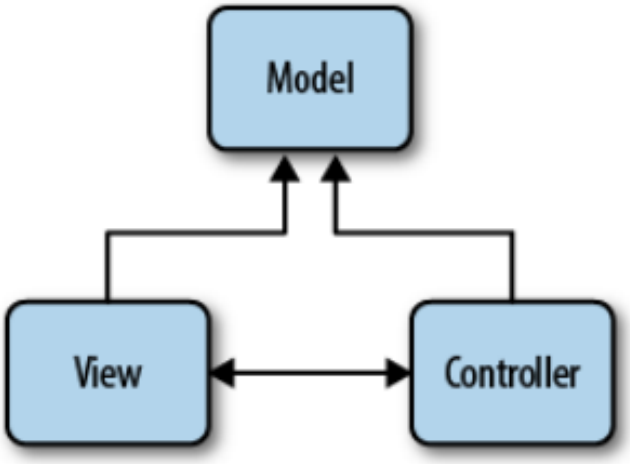


Figura 2.10: Arquitectura Model-View-Controller

En cuanto a los componentes de Spring MVC, este *framework* dispone de los que resultan esenciales en una arquitectura MVC: modelo, vista o controlador. Aunque el rasgo más característico es que dispone de uno adicional llamado DispatcherServlet, universalmente conocido cómo *Front Controller*. Este componente actúa cómo controlador de controladores y se encarga de librar la petición recibida al controlador apropiado para que sea procesada cómo se puede apreciar en la Figura 2.11.

El orden de los acontecimientos en el diagrama de la Figura 2.11 es el siguiente:

1. Al lanzar una petición(generalmente desde un navegador) esta llega al *Front Controller* o DispatcherServlet. Así pués, este componente actúa cómo el punto central de acceso a la aplicación.

2. El *Front Controller* determina cuál es el controlador adecuado para manejar la petición y se la transfiere.

3. El controlador actualiza el modelo de datos(si se requiere) y devuelve el nombre de la vista y el modelo a actualizado para la respuesta al *Front Controller*.

4. El *Front Controller* decide cuál es la vista mas apropiada a utilizar y le transfiere el modelo a esta.

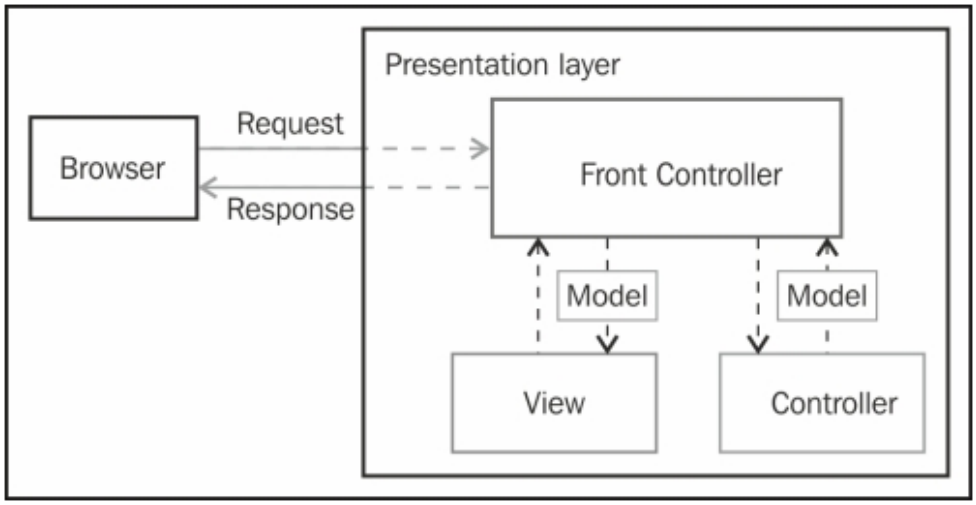


Figura 2.11: Flujo de una petición en SpringMVC

5. La vista incrusta los datos del modelo, los renderiza para obtener la vista web y esta se transfiere al *Front Controller*.

6. Para terminar, el *Front Controller* devuelve la página resultante ya renderizada cómo respuesta al navegador

Características:

Las características principales que ofrece Spring MVC són las siguientes:

- Separación de roles: Dejando de lado los componentes modelo, vista y controlador se pueden añadir más componentes como por ejemplo uno para la seguridad y, además, con la particularidad de que este no influya en el resto.

- Permite configurar todos los objetos ya sean del *framework* o de la propia aplicación cómo si fueran JavaBeans, y es una técnica que consiste en englobar varios objetos como si de uno se tratara para evitar manejar múltiples objetos más simples.

- Permite la utilización de *tags* con el fin de otorgar propiedades predefinidas a los objetos sin necesidad de programarlo.

- Reutilización de código: Por ejemplo, a partir de la definición se puede crear un formulario sin tener que crear una clase con el mismo fin.
- Transferencia de modelo sencilla: Transfiere el modelo en formato clave/valor en las respuestas, por lo cual resulta fácil integrarlo en cualquier tecnología de vistas.
- Obtiene automáticamente parámetros del usuario tales como la zona horaria o el idioma sin necesidad de desarrollar un módulo para ello.
- Permite la definición de *tags* propios de la aplicación, para insertarlos en el código HTML de respuesta.

2.2.3. Spring DataREST

Spring DataREST corresponde a otro módulo bajo el *framework* Spring. Este se encarga de analizar el dominio de datos de la aplicación y proporcionar recursos HTTP para que dicho dominio sea explotado. El objetivo de este tipo de práctica consiste en ofrecer los datos de una forma genérica y que puedan ser integrados por distintos sistemas, ya sea un dispositivo móvil, una aplicación de escritorio u otra aplicación web.

Componentes y arquitectura

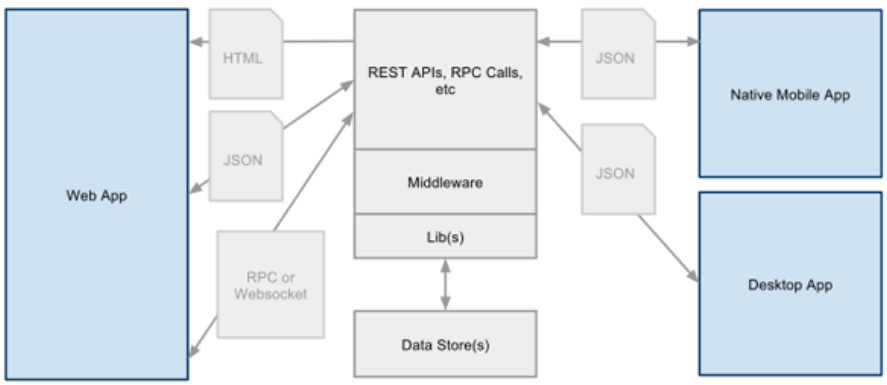


Figura 2.12: Posibles aplicaciones de Spring DataREST

En el diagrama de la Figura 2.12 se muestra una arquitectura genérica para sitios web REST compuesta por los siguientes elementos:

- **Panel central:** servicio web REST o *back-end*.
  - ***Data Store(s):*** corresponde al dominio y persistencia de la aplicación.
  - ***Lib(s):*** corresponde al *framework* Spring.
  - ***Middleware:*** corresponde al controlador de Spring DataREST.
  - **REST APIs, RPC Calls, etc:** corresponde al servicio web que ofrece el *back-end*.
- **Panel intermedio:** corresponde a los diferentes formatos para la transferencia de datos del servicio web a las plataformas que interactúan con este.
- **Paneles laterales:** son las posibles plataformas que pueden consumir los datos ofrecidos por el servicio REST.

Características

Las características principales que ofrece Spring DataREST són las siguientes:

- Proporciona una API REST para el dominio en formato HAL[1](*Hypertext Application Language*).
- Representa el modelo de datos mediante colecciones, objetos y asociaciones entre ellos.
- Soporta la paginación mediante links.
- Permite filtrar dinámicamente las colecciones de recursos.
- Permite manejar las peticiones REST mediante eventos sobre los datos.
- Permite la definición de proyecciones para alterar el formato en el que se devuelven los datos.

[1] Convención estándar para definir hipermedia tales cómo enlaces a recursos dentro de ficheros JSON o XML

- Soporta la integración de distintos tipos y fabricantes de bases de datos.
- Permite customizar los recursos que se ofrecen en el servicio web REST.

2.2.4. Django

Django[15] es un *framework* desarrollado con Python que ofrece un entorno de desarrollo de aplicaciones web. Su objetivo es que bajo este *framework* las aplicaciones se puedan desarrollar rápidamente y sean seguras a la par que escalables.

Componentes y Arquitectura

Django ofrece la posibilidad de crear una aplicación web que cumpla el patrón de diseño MVC cómo el de la Figura 2.10 u ofrecer un servicio web REST y por lo tanto una arquitectura cómo la de la Figura 2.12.

Características

- Django puede generar una API REST automáticamente basada en el modelo sin código adicional.
- Crea y mantiene las tablas dónde se guardarán los datos del modelo automáticamente.
- Permite la herencia de fragmentos de vistas para evitar la redundancia y duplicación de código.
- Control de versiones para la base de datos por si se ve afectada ante cambios en el modelo.
- Incorpora un módulo de seguridad para prevenir ataques.

2.2.5. Symfony

También basado en el patrón de diseño MVC, disponemos del *framework* Symfony[14] desarrollado con PHP. Se caracteriza por proporcionar una aplicación web prediseñada. La única tarea que debe desempeñar el desarrollador consiste en parametrizar y desarrollar la lógica de negocio y las partes no convencionales.

Componentes y Arquitectura:

Aunque se puedan añadir componentes extra desarrollados por terceros, a continuación se detallan los que proporciona Symfony para cumplir con el patrón de diseño MVC:

- **HttpFoundation:** Contiene las clases para manejar las peticiones, ya sea de solicitud o de respuesta.
- **Routing:** Corresponde al nombre que establece Symfony para el controlador.
- **Form:** Herramienta para la creación de formularios.
- **Validator:** Componente que se encarga de comprobar que el tipo y formato de los datos manejados coincidan con los del modelo en caso de interactuar con este.
- **Templating:** Componente para el renderizado de plantillas para las vistas.
- **Security:** Corresponde al componente que establece y garantiza que se cumplan los requisitos de seguridad durante la interacción con la aplicación.
- **Translation:** Componente que asigna una plantilla de idioma a cada vista.

En cuanto a la arquitectura, resulta muy parecida a la de los demás *frameworks* bajo MVC y se representa en el diagrama de la Figura 2.13:

Características

- Compatible con la mayoria de plataformas.
- Dispone de una capa de abstracción que le proporciona compatibilidad con múltiples sistemas de bases de datos.
- Permite incorporar componentes desarrollados por terceros.
- Los formularios que auto-genera incorporan sistemas de validación generados por el *framework* en base al tipo de datos del modelo.
- Las URL que genera para cada recurso resultan amigables y legibles para los usuarios.

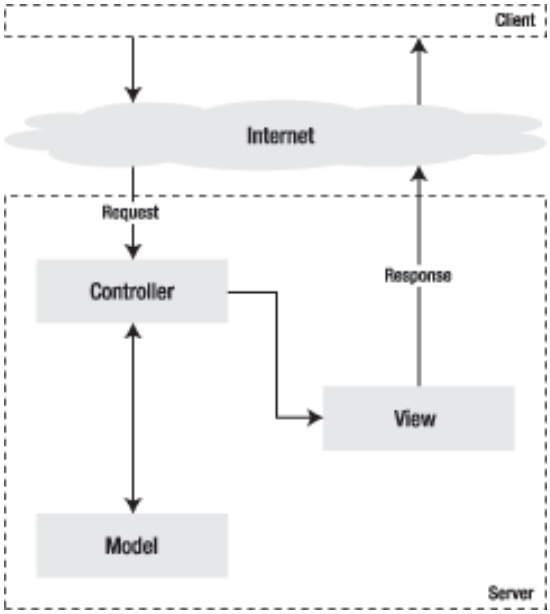


Figura 2.13: Arquitectura MVC de Symfony

- Soporta autenticación y credenciales por lo que se puede restringir el acceso a ciertas áreas.
- Dispone de manejo de cache automático para reducir el uso de ancho de banda y la carga del servidor.

2.2.6. Express

Express[18] es un *framework* desarrollado con NodeJS que ofrece un entorno de desarrollo de aplicaciones web. La propiedad que más le caracteriza es que proporciona un esqueleto básico pero funcional, y al que se le añaden librerías, paquetes y componentes a medida que se requieren.

En otros *frameworks*, los primeros pasos consisten en establecer las dependencias resultando ser un trabajo muy tedioso, además, puede ser que estas varíen durante el desarrollo del proyecto. Mientras que en Express se añaden al proyecto sobre la marcha, resultando ser el rasgo más característico de este *framework*.

Componentes y Arquitectura

Debido a que Express puede adoptar diferentes tipos de arquitectura(MVC, API REST, etc) según los módulos que se le añadan, en la documentación del *framework* no se dispone de ningún apartado que la describa.

Características

Por el mismo motivo que en el apartado anterior, las características reales vienen dadas por los módulos que se añadan al proyecto. Aún así, en la documentación oficial se hace hincapié en las siguientes características no muy técnicas:

- Admite la adición de módulos durante el transcurso del proyecto.
- La aplicación realizada bajo este *framework* será mas ligera que la realizada bajo otros *frameworks* ya que no incluye ningún componente que no utilice.
- Su velocidad de ejecución es superior a la de otros *frameworks* del mismo ámbito.
- La comunidad entorno al *framework* es muy extensa, por lo que se ve reflejado en la cantidad de documentación que existe sobre él.

2.3. *Front-end*

El *front-end* es el componente de la aplicación que ofrece al usuario una interfaz de interacción para manejar la aplicación. Además se encargará de mantener la comunicación con el *back-end* con el fin de obtener o transmitir cambios en los datos del modelo cuando sea necesario.

A diferencia del *back-end*, el código fuente del *front-end* será ejecutado por un navegador web. Hoy en día los navegadores sólo soportan la ejecución de código fuente JavaScript, correspondiente al lenguaje en el que se basa el ya mencionado *framework* NodeJS para el desarrollo de *back-ends*.

De la misma manera que en los *back-ends*, generalmente, para el desarrollo de *front-ends* se utilizan *frameworks* para aprovechar la funcionalidad y arquitecturas que ofrecen en lugar de desarrollar todo el

componente desde cero. Así pues, a priori ya se conoce que todos los *frameworks* de esta sección estarán basados en el lenguaje de programación JavaScript aunque cada uno de ellos ofrece características, arquitecturas y funcionalidades distintas encaradas a diferentes tipos de proyectos.

Por lo tanto, en los siguientes apartados se detallarán los *frameworks* disponibles para el *front-end* más utilizados actualmente(Angular, Ember, React y Polymer), y que además están publicados bajo una licencia libre.

2.3.1. Angular

El primer framework analizado corresponde a Angular[1], actualmente en su versión 4 fue desarrollado y publicado por un equipo de Google bajo la licencia del MIT(*Massachusetts Institute of Technology*), y por lo tanto abierta. Puede resultar confundido con el *framework* AngularJS ya que ambos llevan el mismo nombre pero se distinguen por la coletilla "JS". Además, Angular es un *restyling* de AngularJS realizado por el mismo equipo de este último, aunque las características principales de Angular son las que marcan la diferencia entre uno y otro.

Componentes y Arquitectura:

Una aplicación web realizada en Angular se estructura bajo una arquitectura modular. Cada uno de los módulos representa una funcionalidad de la aplicación de manera independiente a los demás. Además dichos módulos pueden contener a otros de manera recursiva, siempre estableciendo un nodo principal o raíz permitiendo, para cada uno de ellos, establecer una granularidad tan fina como se desee.

Por lo tanto, la arquitectura reflejada en la Figura 2.14 es utilizada por todos y cada uno de los módulos de la aplicación de manera recursiva, des del nodo raíz hasta el último de la jerarquía.

- **Modules:** Cada uno de los submódulos que incluye el módulo en cuestión.

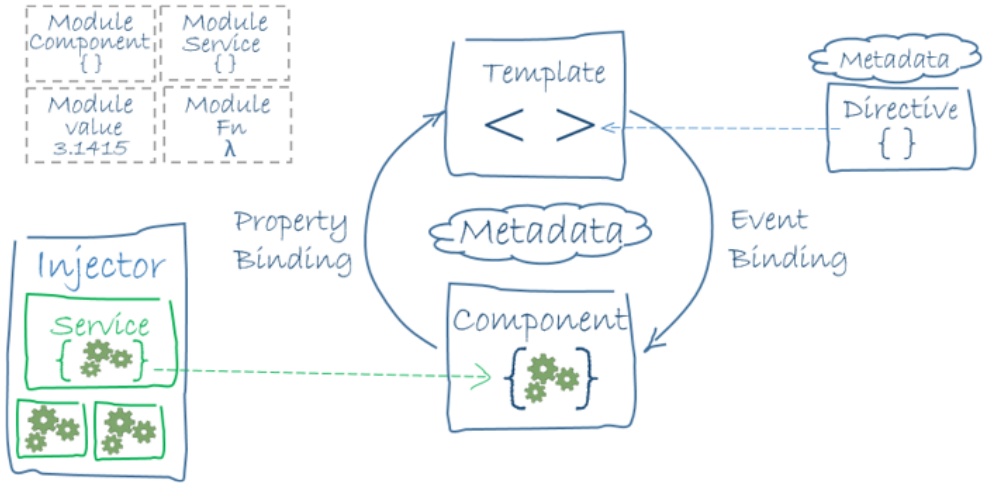


Figura 2.14: Arquitectura de componentes de Angular

- **Template:** La *template* corresponde a la plantilla, la cual una vez renderizada con los datos a mostrar se presentará en el navegador.
- **Metadata:** Se encarga de definir como serán tratados los datos a utilizar por el módulo.
- **Directives:** Sirven para alterar o "programar" una *template* para que se comporte de manera diferente en base a los datos a mostrar.
- **Services:** Constituye la fuente de datos para el módulo y además se encarga de transmitir los cambios realizados en dichos datos.
- **Injector:** Se encarga de la inyección de los *services* necesarios y requeridos para el funcionamiento del módulo.
- **Component:** És el encargado de unificar los componentes descritos anteriormente y formar el módulo en si.

Características:

- El desarrollo del código fuente bajo este *framework* se puede realizar mediante JavaScript o TypeScript. Se aconseja utilizar TypeScript ya que es un supertipo de JavaScript y por lo tanto incluye todas las funcionalidades de este último además de las que aporta el propio lenguaje.

- Permite la reutilización de módulos con tan sólo una línea de código fuente para que lo incluya y, por lo tanto, evita a gran escala la repetición de código.
- Este *framework* está orientado al desarrollo de aplicaciones de cualquier envergadura y naturaleza, de la mas simple hasta la mas compleja.
- Permite el desarrollo de aplicaciones tipo SPA(*Single Page Application*) de manera que ante un cambio o acción sólo será actualizada la parte afectada de la vista(correspondiente a un componente) como si de una aplicación de escritorio se tratara. Por lo tanto se reduce el tráfico y el trabajo que genera el navegador.
- Dispone de la mayor comunidad de desarrolladores de entre todos los *frameworks* para *front-end*.
- Renderizado en la parte del cliente, por lo que disminuye el trabajo realizado por el servidor.

2.3.2. Ember

El framework Ember[2], cuya ultima versión disponible es la 2.13, fue desarrollado y publicado por el equipo Ember Core Team en 2011 bajo la licencia del MIT. Su código fuente está escrito en JavaScript. Su rasgo más característico consiste en que sigue el paradigma CoC(*Convention over Configuration*) cuyo objetivo es decrementar el número de decisiones que ha de tomar el desarrollador sin que la aplicación pierda flexibilidad. Dicho paradigma consiste en que el framework sea quien tome decisiones en los contextos que lo permitan en base a las convenciones, de manera que el desarrollador solo tendrá que intervenir en caso que esté en desacuerdo con ello.

Arquitectura y Componentes:

Como se puede apreciar en la Figura 2.15, el fabricante proporciona un diagrama con un ejemplo de cómo se maneja una petición a través de sus componentes para exponer la arquitectura.

- ***Router:*** También llamado controlador, se encarga de redirigir la petición al *Route Handler* en base a algún parámetro de esta.

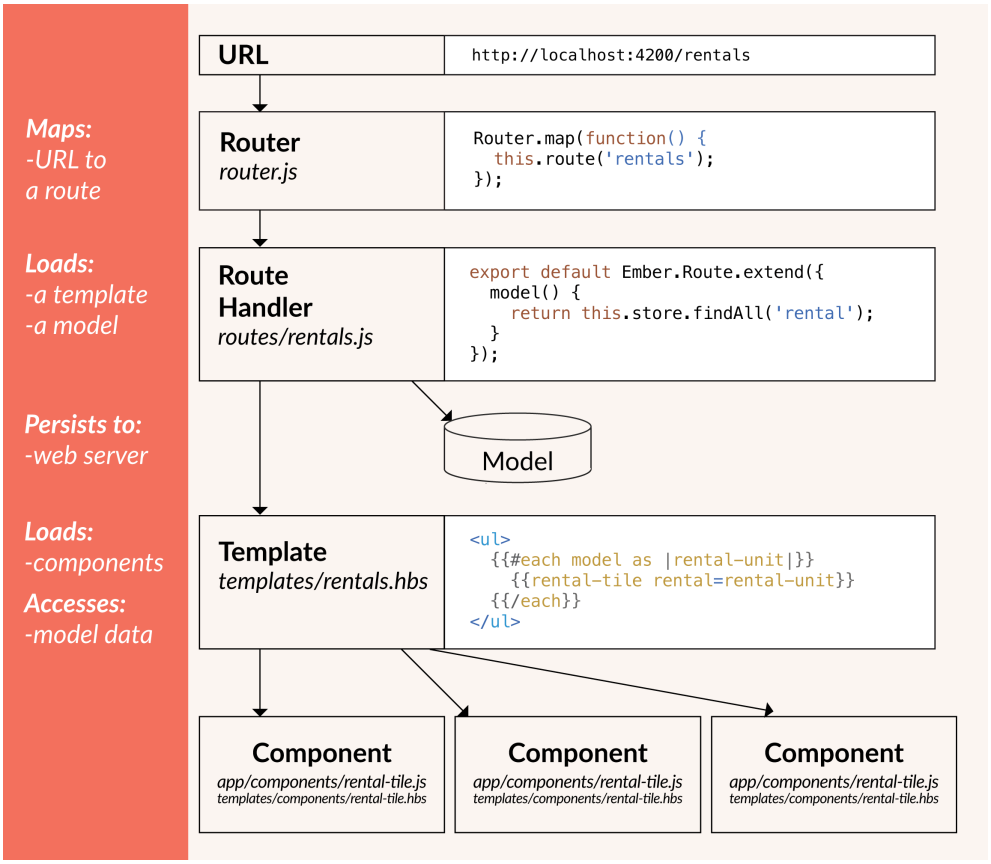


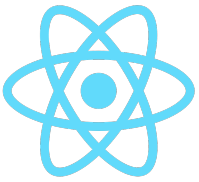
Figura 2.15: Arquitectura de componentes de Ember

- ***Route Handler:*** Recibe la petición del *Router* y carga el modelo de datos(*Model*) y la *Template* a utilizar.
- ***Model:*** Contiene los datos o registros a mostrar en las vistas.
- ***Template:*** Se encarga de la definición de las vistas, las cuales renderizadas con los datos del *Model* se devolverán al usuario como respuesta a la petición.
- ***Component:*** Se encarga de definir el comportamiento de los elementos de las *Templates* una vez recibidas por el usuario, como por ejemplo las acciones que realizarán los botones.

Características:

- De la misma manera que Angular, permite el desarrollo con TypeScript, además de ser renderizado en la parte del cliente.
- Utiliza el paradigma CoC ya mencionado.
- Permite interfaces de usuario anidadas.
- Utiliza observadores y solo vuelve a renderizar los valores alterados en vez de la vista completa.
- Dispone de una buena documentación.
- Tiempos de arranque y estabilidad inherente.

2.3.3. React



React[9] fue desarrollado y publicado por la comunidad de Facebook e Instagram en 2013 bajo la licencia del BSD-3(*Berkeley Software Distribution*) y por lo tanto libre. La última versión de su código fuente corresponde a la 15.5 y está escrito en JavaScript. Su rasgo más característico es que resulta tan ligero como una librería, aunque tan potente que se le considera un framework. Está enfocado a la creación de interfaces que deban mostrar gran cantidad de registros del mismo tipo, aunque con una lógica sencilla. También permite ser integrado en otros frameworks como los ya mencionados Angular y Ember.

Características:

- React genera las vistas a mediante instrucciones JavaScript con el código que define vista, como por ejemplo HTML.
- Ante un cambio en el modelo de datos permite la actualización y re-renderizado de elementos de la vista con una granularidad muy fina.
- Permite realizar interfaces nativas para dispositivos móviles.
- Mantiene una copia virtual de los datos en memoria y solo actualiza aquellos que hayan sido alterados, por lo que agiliza las operaciones donde estos se vean involucrados.

- Permite ser integrado en otros proyectos con el fin de generar las vistas.
- Debido a que puede ser integrado en proyectos, puede ser utilizado en el lado del cliente así como en el del servidor, siempre y cuando este último utilice el patrón MVC.

2.3.4. Polymer

Polymer[8] fue desarrollado y publicado por Google en 2015 bajo la licencia del BSD-3 y por lo tanto libre. Su versión más reciente de código fuente corresponde a versión 1.9.1 y está escrito en JavaScript, además también resulta ser una librería, como React. Su cometido es ofrecer funcionalidades aún por estandarizar por el W3C(*World Wide Web Consortium*) y que en un futuro serán incorporadas de manera nativa en los navegadores. Por lo tanto, esta librería verá decrementadas sus funcionalidades a medida que sean añadidas por defecto en los navegadores. Además incorpora estilos y plantillas para que las interfaces realizadas bajo esta librería adopten el estilo *Material Design* de Google.

Características:

- Permite la definición de elementos personalizados equivalentes a elementos de las vistas. Estos pueden ser almacenados en repositorios comunitarios y ser utilizados por terceros, de la misma manera que se pueden utilizar en proyectos propios los desarrollados por otros desarrolladores.
- Permite la definición de *listeners* y manejar los eventos realizados por estos.
- De manera opcional, se puede utilizar una librería para manejar eventos gestuales.
- Permite enlazar elementos de las vistas con atributos de los elementos del modelo de datos.

Capítulo 3

Diseño

La tarea de diseño consiste en realizar los esquemas y diagramas requeridos para la siguiente fase, la de implementación, en base a los resultados obtenidos en el estudio sobre las distintas tecnologías en la fase de análisis. De este modo, se optará por las que mayor ventaja y utilidad aporten para la realización del proyecto.

3.1. Comunicación con el equipamiento

En la fase de diseño de la comunicación con el equipamiento se expondrá la tecnología escogida a utilizar para el desarrollo del proyecto. Dicha tecnología será la que mayor ventajas aporte en la realización del proyecto en base a los resultados del estudio realizado durante la etapa de análisis de este ámbito. Además se incluirán los diagramas y esquemas requeridos en los que se basará el desarrollo de este componente durante la fase de implementación del proyecto.

El abanico de tecnologías que ofrece el equipamiento para ser gestionado de manera remota, las cuales fueron analizadas en la fase de análisis de este mismo ámbito son las siguientes:

- El protocolo SNMP.
- La Utilización de *WebServices* mediante MIBs.
- La Utilización de *WebServices* mediante CLI.

Para este proyecto se ha optado por la utilización de ***WebServices* mediante CLI** debido a las ventajas que ofrece sobre el resto. En los dos siguientes apartados se listan las ventajas que ofrece el mecanismo

escogido sobre los demás. En el primero se comparará los *WebServices* frente SNMP y en el segundo dichos *WebServices* mediante CLI frente la variante con MIBs.

3.1.1. *Webservices* CLI frente el resto de opciones

SNMP:

1. SNMP requiere de una configuración inicial más compleja frente a los *WebServices* que además vienen activados por defecto.

2. SNMP solo permite consultar y modificar valores a nivel de MIBs con las desventajas que eso conlleva: realizar un estudio previo de las MIBs afectadas en un cambio en la configuración y conocer el formato de los valores que alberga cada una de ellas ya que pueden incluso representarse como tablas. Mientras que los *WebServices* permiten distintos modos de gestión de la configuración más sencillos.

3. SNMP del mismo modo que los *WebServices* funciona mediante peticiones, aunque las de este ultimo suelen ser menos complejas. Además dichas peticiones se realizan bajo el protocolo HTTP que resulta ser más utilizado y por lo tanto el volumen de documentación y herramientas para su implementación y testeo es destacable.

***WebServices* mediante MIBs:**

1. La variante con MIBs conlleva las mismas desventajas que el segundo punto del apartado anterior. Mientras que con CLI sólo se requiere del comando que sería utilizado en el terminal del equipo, y este último se encarga de sintetizar-lo y aplicar los cambios de configuración pertinentes en todos los niveles de parámetros del dispositivo, incluidas las MIBs.

2. Mediante MIBs hay que utilizar un método HTTP diferente según se obtenga, se actualice, se añadan o se eliminen valores mediante los métodos GET, PUT, POST, DELETE respectivamente. Mientras que mediante CLI sólo hay que utilizar el método GET para cualquier tipo de operación.

Una vez justificada la toma de decisiones , se detallará el funcionamiento de los *WebServices* mediante CLI.

Funcionamiento

En el momento que devienen activos los *WebServices*, el equipamiento actúa como un servidor web el cual recibe y resuelve peticiones HTTP de los clientes adoptando una arquitectura cliente-servidor.

Según las especificaciones de la documentación oficial, la realización de una o varias peticiones de un cliente cualquiera al equipamiento que actúa como servidor debería seguir el flujo reflejado en el diagrama de secuencia de la Figura 3.1.

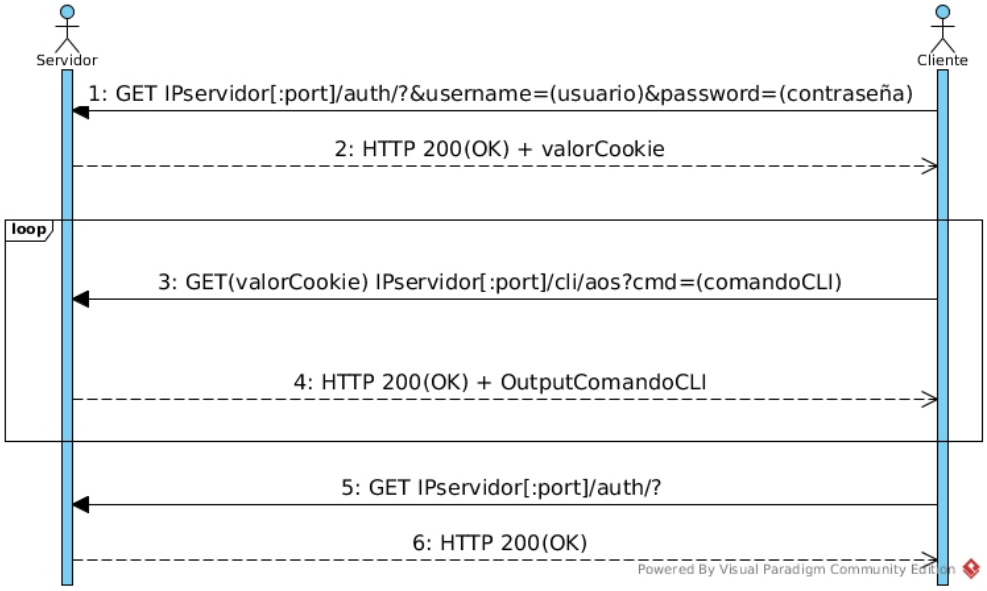


Figura 3.1: Diagrama de secuencia de la comunicación con el equipamiento

A continuación se detalla el objetivo de cada una de las peticiones y respuestas. Además se da por sentado que el tipo de peticiones HTTP que realiza el cliente son mediante el método GET:

1. El cliente realiza el paso de autenticación al servidor.
2. El servidor responde con un código HTTP 200(OK) e incluye la *cookie* de sesión en la respuesta.
3. El cliente realiza la petición y se añade la *cookie* recibida en el punto anterior.

4. El servidor responde con un código HTTP 200(OK) e incluye la salida generada por el comando CLI de la anterior petición.
5. El cliente solicita des-autenticarse.
6. El servidor responde con un código HTTP 200(OK).

3.2. *Back-end*

Una vez resuelto el diseño de la comunicación con el equipamiento en el apartado anterior, se procederá a realizar la misma tarea para el siguiente componente: el *Back-end*.

Así pues, según vimos en la respectiva fase de análisis las tecnologías disponibles y analizadas para el desarrollo de este componente son:

- .NET
- Spring MVC
- Spring DataREST
- Django
- Symfony
- Express

En base a las ventajas e inconvenientes de cada una de ellas, se ha optado por la realización del proyecto mediante el *framework* **Spring DataREST**, pues es el que ofrece una relación ventajas-inconvenientes más beneficiosa para el proyecto. Para justificar dicha decisión se expondrá en los siguientes apartados los beneficios que aporta la utilización del *framework* por el que se ha optado frente al resto, teniendo en cuenta que .NET se puede despreciar ya que no cumple con los requisitos básicos de ofrecer una licencia libre: és propietario de Microsoft.

3.2.1. Spring DataREST frente el resto de *frameworks*

Spring MVC

La principal ventaja de Spring DataREST sobre Spring MVC és que el primero de ellos ofrece una API REST, la cuál permite el alta,

baja y modificación de datos de manera genérica entre otros bajo el estándar REST. Además dichos datos se ofrecen en formatos genéricos como pueden ser JSON o XML, los cuales pueden ser consumidos por cualquier cliente que permita realizar peticiones HTTP. De modo que se pueden generar *front-ends* para cualquier arquitectura sin que la fuente de datos requiera ser adaptada a cada una de ellas como por ejemplo: una aplicación web, una aplicación de escritorio, una aplicación nativa para Android o una aplicación nativa para iPhone entre otros.

Spring MVC, como su propio nombre indica, ofrece una arquitectura bajo el patrón MVC. Este patrón ofrece menos flexibilidad debido al tipo de vistas web que genera. Dichas vistas ante cada acción realizada por el usuario, requiere que sea actualizada la totalidad de la página, en vez de únicamente la parte afectada. Este patrón se empieza a considerar obsoleto y ha sido reemplazado por el ya nombrado Single Page Application, que ofrece una mejor experiencia de usuario muy parecida a la de una aplicación de escritorio. Para ello se requiere una fuente de datos tipo Spring DataREST y un *front-end* encargado de manejar los datos y mostrarlos en las vistas que este último genera.

Django

Cabe destacar que ambos *frameworks* ofrecen ventajas muy parecidas, ya que permiten la creación de una aplicación web que ofrezca una arquitectura y funcionalidad REST, además dicha API REST se genera automáticamente en base al modelo de datos definido. Por lo tanto, el balance de ventajas e inconvenientes se realizará a partir de factores externos a los *frameworks* así como el volumen de información relativa a ellos, el tamaño de la comunidad que lo utiliza y la calidad de la documentación que ofrecen.

Para resolver dicho balance, se ha recurrido a la gráfica obtenida en [13] que muestra la Figura 3.2, la cuál refleja el índice de popularidad de cada uno de los lenguajes de programación frente al resto. Dicho índice se ha calculado en base a: el número de ingenieros cualificados en todo el mundo, el numero de cursos disponibles y terceros que ganan dinero a partir de dicho lenguaje además del volumen de la información manejada relativa a cada lenguaje en los buscadores más importantes(Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube y Baidu).

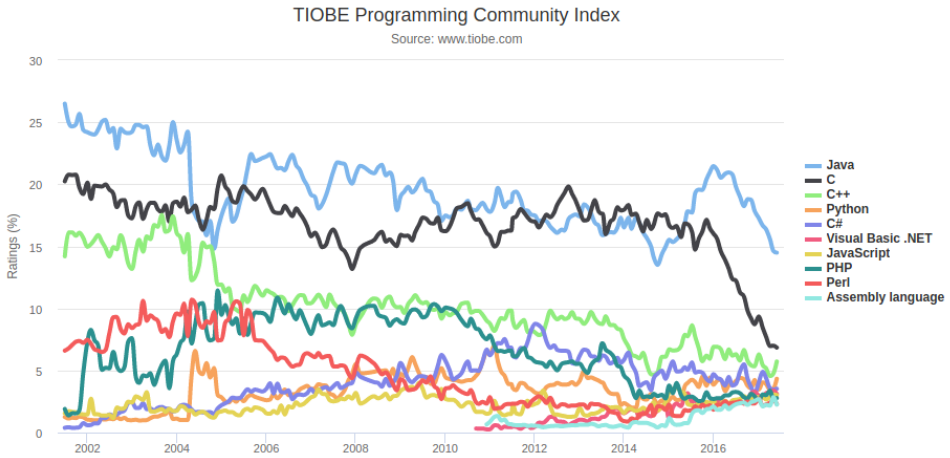


Figura 3.2: Índice de popularidad de los principales lenguajes

Dichos resultados están liderados por Java, en el cual está basado el *framework* Spring y por ende, Spring DataREST. De esta manera quedan resueltas las dos primeras cuestiones: el volumen de información relativa a ellos y el tamaño de la comunidad que lo utiliza. En cuanto a la tercera, queda resuelta a partir de la respuesta a las dos primeras teniendo en cuenta que cualquier punto no resuelto por la documentación oficial, puede encontrarse en la documentación de terceros.

Symfony

La arquitectura por defecto que Symfony propone corresponde a MVC para las desarrolladas bajo este *framework*, aunque mediante la adición de componentes desarrollados por terceros también puede ofrecer una arquitectura REST. Aún así, resulta preferible utilizar un *framework* con la arquitectura REST de manera nativa como lo ofrece Spring Data-REST.

Además, también resultan aplicables en este contexto las ventajas que ofrece Spring DataREST en los dos apartados previos en los cuales se ha justificado el motivo por el cual se ha optado por este *framework*.

Express

En primer lugar, como ya se argumentó en la fase de análisis, Express no dispone de ninguna arquitectura por defecto, ya que esta viene definida por los módulos que se le añadan. A partir de aquí, puede ofrecer cualquier tipo de arquitectura, incluida REST. Aún así no existe ninguna que permita generar de manera automática una aplicación REST en base al modelo de datos definido en él, además, existen diferentes módulos con la misma finalidad, por lo que esto disminuye el volumen de información que existe para cada una tarea en concreto.

Así pues, una vez justificada la decisión de optar por Spring DataREST para el desarrollo del proyecto en el ámbito del *back-end*, se presentarán los diagramas necesarios para llevar a cabo la implementación de este componente en los siguientes apartados.

3.2.2. Estructura

El *back-end* se verá involucrado en dos arquitecturas cliente-servidor. En la primera de ellas el equipamiento actuará cómo servidor mientras el *back-end* lo hará cómo cliente. Para la segunda, el *back-end* tendrá el rol de servidor mientras la API REST que ofrece será explotada por cualquier cliente. Por lo tanto el *back-end* actuará cómo intermediario entre el usuario final y el equipamiento cómo se puede apreciar en la Figura 3.3.

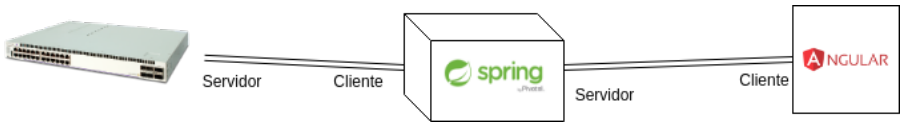


Figura 3.3: Estructura equipamiento/back-end/clientes

Persisténcia

Por otra parte, el *back-end* también se encargará de la persistencia de datos, la cuál mediante una base de datos que permite almacenar datos de un entorno real en un entorno virtual. En este caso para gestionar

lo referente al patrimonio arquitectónico y equipamiento tecnológico como se mencionó en la introducción del documento.

Genéricamente se consideran dos tipos de bases de datos: las relacionales(SQL) y las no relacionales(NoSQL). Según [25], el primer tipo ofrece esquemas bien definidos y permite relaciones muy dependientes. Mientras que NoSQL ofrece justamente lo contrario, más bien maneja mejor el desorden.

A partir de las afirmaciones anteriores, se ha optado por la utilización de una base de datos relacional ya que cada registro de cada entidad requiere el mismo numero de campos y relaciones, además estas últimas se han de garantizar.

Una vez se ha optado por el tipo de base de datos también se hace necesario escoger un sistema gestor de base de datos, que será el encargado de manejarla internamente. Las principales opciones con licencia libre son: MySQL, MariaDB y PostgreSQL. Debido a que el foco del proyecto no es el almacenamiento de datos de una base de datos no se ha realizado ningún estudio previo sobre cuál de ellas es la mejor opción. Por lo tanto se ha optado por la utilización de **PostgreSQL** debido a que es con la que se dispone de más bagaje y además, se conoce a priori que cumple con los requisitos del proyecto y es compatible con Spring.

Una vez escogido el sistema gestor de la base de datos se propone el esquema relacional de la Figura 3.4 para su definición.

Entidades:

- ***Campus:*** Define la entidad que representa un conjunto de edificios.
- ***Building:*** Entidad que representa un edificio.
- ***Floor:*** Entidad que representa cada una de las plantas que contiene un edificio.
- ***Connector:*** Representa cada uno de los puntos de red disponibles en cada sala.
- ***EquipmentRoom:*** Representa una sala destinada a albergar equipamiento
- ***Equipment:*** Representa cada uno de los equipos a gestionar.

- ***Card:*** Representa cada placa que forma el dispositivo.
- ***Port:*** Representa los puertos de los que dispone cada una de las tarjetas.

Relaciones:

- ***Campus - formedBy(oneToMany) - Building:*** Representa la relación entre los edificios que forman un Campus. Un campus puede contener varios edificios, pero un edificio sólo puede estar contenido por un Campus.

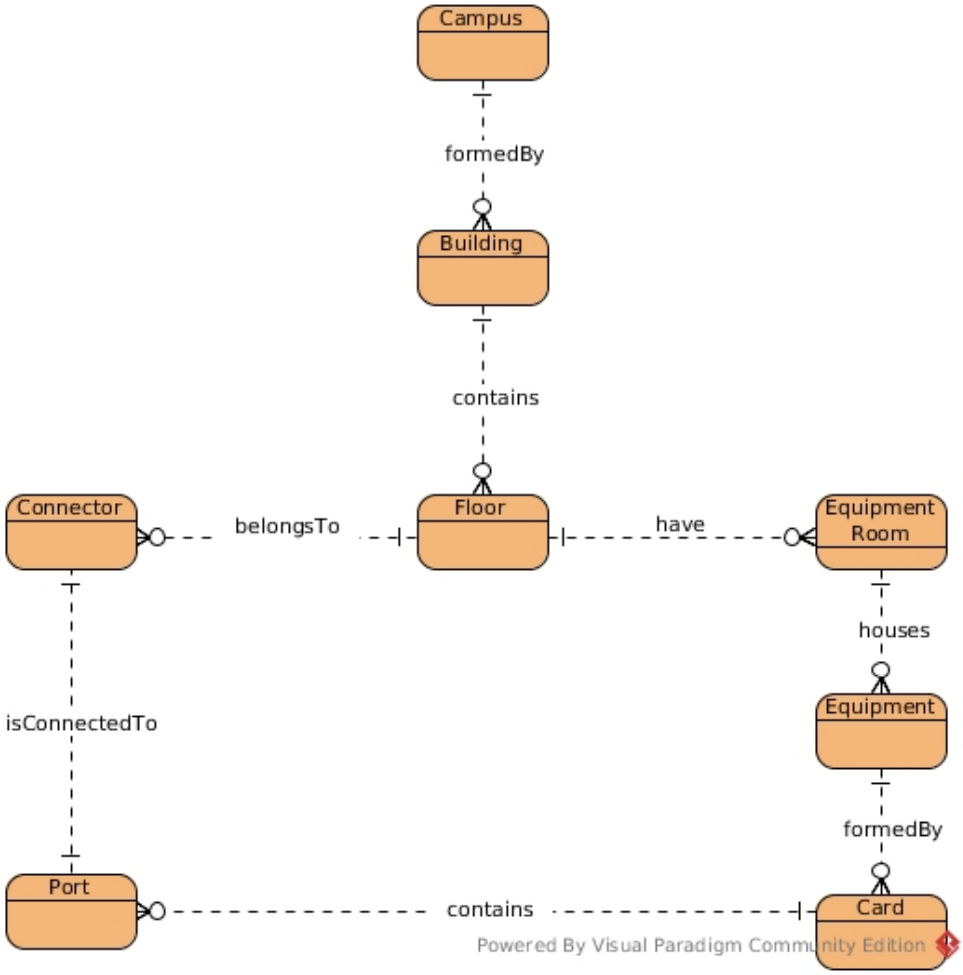


Figura 3.4: Esquema relacional de la base de datos

- ***Building - contains(oneToMany) - Floor:*** Representa la relación entre las plantas que contiene cada uno de los edificios. Un edificio puede contener diferentes pisos, pero cada piso sólo puede ser contenido en un edificio.
- ***Floor - belongsTo(oneToMany) - Connector:*** Representa la relación entre cada piso y los conectores que contiene. Un piso puede contener varios conectores, pero cada conector puede pertenecer únicamente a un piso.
- ***Connector - formedBy(oneToOne) - Port:*** Representa la relación entre cada uno de los conectores con cada uno de los puertos que ofrecen las tarjetas de los equipos. Relación exclusiva uno-a-uno, cada conector únicamente corresponde a un puerto y viceversa.
- ***Floor - have(oneToMany) - EquipmentRoom:*** Representa la relación entre las salas de equipamiento que tiene cada uno de los pisos. Cada piso puede contener varias salas de equipamiento, pero estas únicamente pueden pertenecer a un piso.
- ***EquipmentRoom - houses(oneToMany) - Equipment:*** Representa la relación entre las salas de equipamiento y el equipamiento que esta alberga. Cada sala puede albergar varios equipos, pero cada equipo únicamente puede ser albergado por una sala.
- ***Equipment - formedBy(oneToMany) - Card:*** Representa la relación entre cada equipo y las tarjetas que lo compone. Cada equipo puede contener varias tarjetas, pero estas sólo pueden encontrarse en un equipo.
- ***Card - contains(oneToMany) - Port:*** Representa la relación entre cada una de las tarjetas y cada uno de los puertos que ofrecen estas. Cada tarjeta puede contener varios puertos, pero cada uno de estos únicamente puede ser ofrecido por una tarjeta.

3.3. *FontEnd*

En esta sección se va a proponer el *framework* a utilizar para el *front-end* en base a los estudiados en la correspondiente fase de análisis. Además se va a justificar dicha decisión exponiendo un balance de ventajas y desventajas que ofrece el *framework* seleccionado frente al resto.

En la fase de análisis se estudiaron los siguientes *frameworks*:

- Angular
- Ember
- React
- Polymer

Finalmente se ha optado por **Angular** debido a las ventajas que ofrece su utilización frente al resto. Dicha decisión se justificará en los próximos apartados mediante una comparación de Angular con cada uno de los demás *frameworks*.

3.3.1. Angular frente el resto de *frameworks*

Ember

Los *frameworks* Angular y Ember están muy a la par en cuanto a características y ventajas que ofrecen. Ambos permiten: una arquitectura SPA, el desarrollo mediante el lenguaje de programación TypeScript para el código fuente con las ventajas que ello conlleva, además de proporcionar de una buena documentación.

A grandes rasgos, el hecho de que se haya optado por Angular se debe a su mayor flexibilidad, a una mejor segregación de componentes muy bien definida y lógica, además de su adaptabilidad a cualquier tipo de proyecto. También cabe destacar que Google es quién está detrás de Angular, manteniendo y desarrollando mejoras para dicho *framework*.

Por último, Angular resulta ser el framework más utilizado actualmente y esto, además de ser un buen indicador, supone disponer de más volumen de información y mas reciente. Dicha afirmación se respalda mediante el gráfico proporcionado por la herramienta Google Trends disponible en [3] y mostrado en la Figura 3.5, en la cual se refleja las búsquedas y la información relativa a los *frameworks* estudiados durante los últimos cinco años, dónde se puede apreciar cómo Angular se ha impuesto sobre el resto por lo que se interpreta que es el más utilizado actualmente.

React

React, como ya se explicó en la correspondiente fase de análisis resulta ser una librería aunque muy potente, aún así no se considera un

Figura 3.5: Impacto de cada uno de los *frameworks* en la herramienta de búsqueda de Google

framework debido a sus carencias en cuanto a herramientas y funcionalidades que incorpora. El hecho de que sea una librería, permite integrarlo en proyectos realizados con *frameworks*, cómo por ejemplo en Angular. Por consiguiente, debido a que dicha integración sólo se puede realizar en un sentido(Angular integra React), resulta mucho más lógico basar el proyecto en Angular y, de este modo, en caso que sea necesario se puede aprovechar el potencial de ambos componentes para un mismo proyecto.

Polymer

Del mismo modo que React, Polymer también se considera una librería y, además, puede ser integrada en un proyecto realizado bajo Angular. De este modo, resulta evidente optar por este último si puede integrar y aprovechar el potencial de distintas librerías las cuales están cerca de ser consideradas *frameworks*.

3.4. Diseño Global

Para concluir este apartado, una vez detallado el diseño de cada uno de los componentes principales, se presentará un diagrama de interacción entre estos últimos para obtener una visión global de la plataforma. Dicho diagrama reflejado en la Figura 3.6, y la función de sus respectivos componentes se desarrollará brevemente en los siguientes apartados.

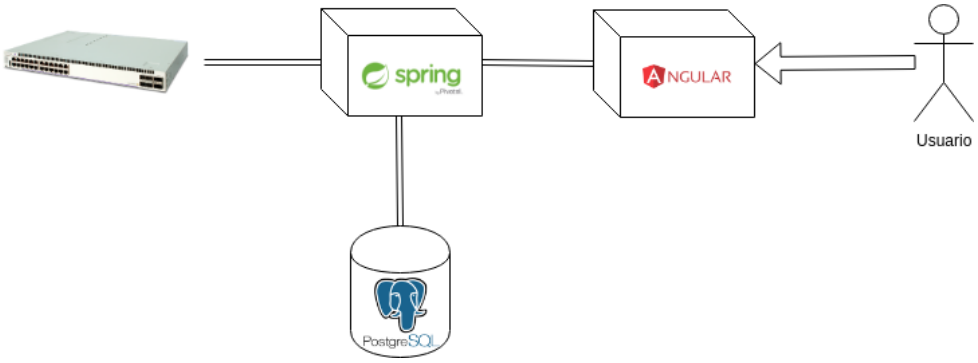


Figura 3.6: Visión global de los componentes de la plataforma

3.4.1. Equipo

El equipo o más bien su configuración resulta ser el objetivo del proyecto. Se encarga de recibir y responder a las peticiones generadas por el *back-end* con el objetivo de:

1. Conocer los valores configurados.
2. Conocer los valores configurables.
3. Aplicar cambios en su configuración en base a los dos anteriores.

3.4.2. Spring

Corresponde al componente denominado *back-end*. Debido a su posicionamiento corresponde al núcleo de la plataforma, ya que es el encargado de enlazar el resto de componentes además de ser el que de más tareas se encarga. Para listar dichas tareas se han segregado en grupos representando cada una de las relaciones de este componente con el resto:

- **Spring - Equipo:**
  1. Consultar los valores configurados en el equipo.
  2. Consultar los valores configurables en el equipo.
  3. Aplicar cambios en la configuración del equipo.
- **Spring - PostgreSQL:**
  1. Definición del modelo de datos.
  2. Alta, baja, y edición de las entidades.
  3. Gestionar las relaciones entre entidades.
- **Spring - Angular:**
  1. Recibir y responder las peticiones de Angular las cuales consisten en: alta, baja y edición de las entidades.
  2. Recibir y responder las peticiones de Angular para: consultar valores configurados y configurables en el equipo además de gestionar los cambios en la configuración.

3.4.3. PostgreSQL

PostgreSQL corresponde a la base de datos encargada de la persistencia de los datos requeridos para el funcionamiento de la plataforma.

Se encarga de almacenar y garantizar las relaciones de los datos referentes a las entidades definidas en el modelo relacional de la Figura 3.4.

3.4.4. Angular

Corresponde al componente denominado *front-end*. Es el intermediario entre el Usuario y Spring. Su tarea consiste básicamente en proveer de una interfaz al usuario para manejar los datos, ya sea de las entidades de la base de datos, o bien de la configuración del equipo.

3.4.5. Usuario

El usuario, mediante el *front-end*, se encarga de inicializar y gestionar los datos para el funcionamiento de la plataforma y de aplicar, a posteriori, los cambios en la configuración de los equipos.

Capítulo 4

Implementación

La implementación corresponde a la fase del proyecto en la que se lleva a cabo el desarrollo de los componentes a partir de los esquemas generados en la fase de diseño.

4.1. Comunicación con el equipamiento

En cuanto al desarrollo de la comunicación con el equipamiento se ven involucrados el equipamiento y el *back-end*. El primero de ellos actúa cómo servidor y el segundo como cliente.

4.1.1. Acondicionamiento del equipamiento

En primer lugar hay que establecer la configuración del equipamiento para que permita la recepción de peticiones HTTP y poder explotar su API.

Dicha configuración se regula mediante comandos CLI del terminal del equipo. Para acceder a dicho terminal, en este caso, se ha utilizado una conexión con el puerto Serie del dispositivo a través de la interfaz microUSB. Mediante un cable microUSB/USB2.0, se ha conectado este ultimo extremo a un puerto USB cualquiera de un PC y se ha lanzado la herramienta Minicom. Resulta importante mencionar que cuando se ejecuta esta herramienta por primera vez también debe ser debidamente configurada para que los parámetros de configuración de la conexión coincidan con los del puerto serie del dispositivo a utilizar. En el caso de los equipos Alcatel utilizados corresponden a los siguientes:

- **Bps/Paridad/Bits:** 9600 8N1

- **Control de Flujo por Hardware:** No
- **Control de Flujo por Software:** No

Para el proyecto se ha partido de un equipo el cual disponía de la configuración por defecto. Por ello se ha realizado previamente un estudio sobre los comandos CLI a ejecutar para que el equipamiento pueda atender a las peticiones web.Y a través de la herramienta Postman de Google Chrome diseñada para el testeo de servidores web, mediante la cual se permite el envío y recepción de peticiones HTTP parametrizadas de cualquier tipo(GET, PUT, POST, etc) se ha comprobado que el equipamiento ofrece el comportamiento deseado y así satisfacer los requerimientos del *back-end* cómo futuro cliente.

Los comandos CLI a ejecutar en los equipos para poder establecer una conexión HTTP satisfactoria son los siguientes:

1. En primer lugar se debe crear una interfaz y asignar a esta una dirección IP dentro de la VLAN1 que los equipos llevan habilitada por defecto mediante el comando CLI:
   `ip interface vlan1 address 192.168.1.1/24 vlan1`

2. En segundo lugar hay que habilitar la autenticación de usuarios al portal web del dispositivo ya que las conexiones HTTP se realizan a través de este. Esta acción se lleva a cabo mediante el comando CLI:
   `aaa authentication http local`

3. En tercer y último lugar hay que reemplazar el protocolo HTTPS por HTTP para simplificar el proceso de establecimiento de conexión a su portal web. Esta acción se lleva a cabo mediante el comando CLI:
   `webview force-ssl disable`

Una vez el equipo ya está preparado para establecer conexiones de red a través de sus puertos, se procede a conectar un PC en uno de estos mediante un cable de red con conectores RJ45 en ambos extremos. Uno de los extremos debe conectarse a un puerto cualquiera del equipo y el otro extremo a la tarjeta de red del PC. Una vez realizada la conexión física se hace necesario configurar la tarjeta de red en este último, pues hay que asignar una dirección IP válida de dentro del rango de la red a la que pertenece la IP asignada en la interfaz de la VLAN1 del equipo.

Por ejemplo: `192.168.1.3/24`.

Para comprobar que la conexión se ha establecido satisfactoriamente se puede lanzar la herramienta `ping <ip dispositivo remoto>` y comprobar que se realiza con éxito. En caso afirmativo ya se puede comenzar a lanzar peticiones HTTP al equipo.

4.2. Comandos CLI a ejecutar

El equipamiento ha de ser capaz de lanzar los comandos CLI incluidos en las peticiones HTTP listados a continuación y organizados según la naturaleza de su cometido:

Conocer los valores configurados

- `show interfaces <chassis>/<slot>/<port>status` mediante el cual se obtienen los parámetros configurados de un puerto.
- `show interfaces <chassis>/<slot>/<port>traffic` mediante el cual se obtienen las estadísticas de tráfico de un puerto.
- `show vlan members <chassis>/<slot>/<port>` mediante el cual se obtiene la VLAN asignada a un puerto.
- `show running-directory` mediante el cual se obtiene el directorio de trabajo actual.

Conocer los valores configurables

- `show interfaces <chassis>/<slot>/<port>capability` mediante el cual se obtienen los parámetros configurables para un puerto.
- `show vlan` mediante el cual se obtiene el listado de VLAN disponibles.

Respaldar la configuración

- `copy certified <directory>make-running-directory` mediante el cual se establece el nuevo directorio de trabajo.
- `write memory` mediante el cual se respalda la configuración actual en el directorio especificado en el comando anterior.

- `copy running certified` mediante el cual se establece que la configuración del directorio actual será la utilizada en el equipamiento tras el próximo reinicio.

4.3. *Back-end*

Una vez configurado el equipamiento para mantener conexiones HTTP procedemos a explotar la API que ofrecen sus servicios web. Dicha API será explotada des del componente que se expone en esta misma sección: el *back-end*.

Este componente, de acuerdo con lo especificado en la fase de diseño, se desarrollará bajo el framework Spring.

4.3.1. Inicialización

Así pues, para el desarrollo de un programa bajo este *framework*, el primer paso a realizar consiste en la inicialización de un proyecto por defecto, el cual incluirá únicamente las posibles dependencias que se requieran durante el desarrollo del mismo.

Spring resulta ser muy susceptible a errores provocados por incompatibilidades entre las versiones de sus dependencias, por lo que se ha recurrido a la herramienta Spring Initializr. Esta herramienta disponible en [11] y mostrada en la Figura 4.1 permite inicializar un proyecto bajo este framework de manera que incluye únicamente los parámetros básicos así como: el nombre, el grupo, la versión de Java a utilizar y el gestor de dependencias entre otros. No obstante lo que la hace mas interesante consiste en un selector de dependencias, de manera que según las que se haya escogido modifica la versión de cada una de estas para que todas sean compatibles de manera conjunta.

Una vez finalizada la introducción de los atributos básicos del proyecto y la selección de dependencias permite descargar el proyecto comprimido para empezar a desarrollar en él.

La lista de dependencias requeridas para este proyecto corresponde a la siguiente:

- **JPA (Java Persistence API):** Permite el manejo de datos relacionales entre las entidades definidas para la aplicación.

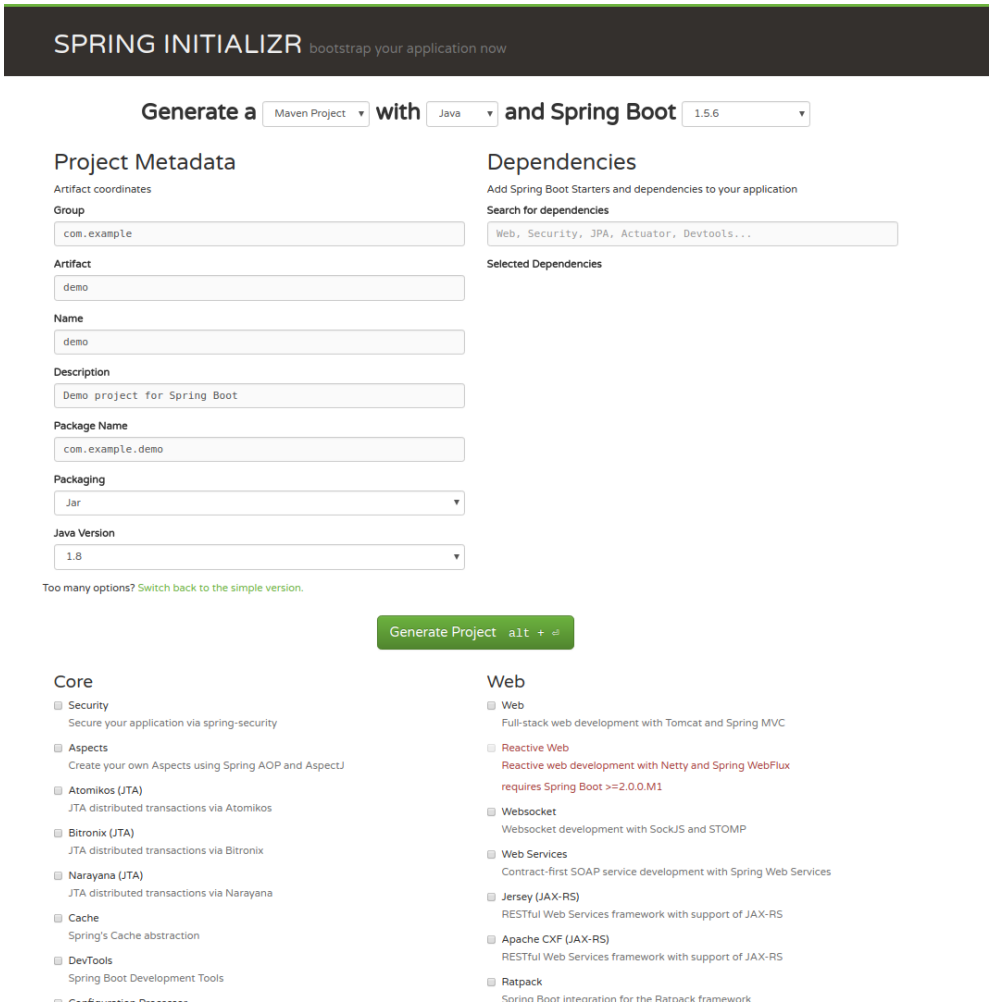


Figura 4.1: Herramienta Spring Initializr

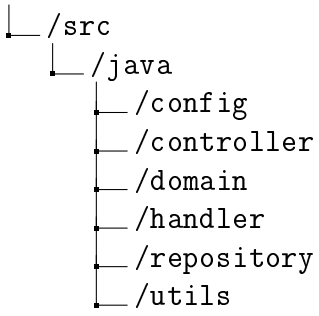
- **REST:** Define la arquitectura y funcionalidad REST en la aplicación.
- **REST Hal Browser:** Proporciona un servicio web adicional el cual permite interactuar con la API REST generada sin tener que utilizar aplicaciones de terceros para el debug o testeo de la misma.
- **HATEOAS:** Incluye un link en los datos de cada entidad para poder navegar por sus entidades adyacentes y, por lo tanto, por todo el modelo de datos mediante links.

- **Security:** Aporta al proyecto mecanismos de autenticación y autorización de usuarios.
- **Validation:** Permite añadir métodos de validación customizados y por lo tanto alternativos a los que se incluyen por defecto. Estos permiten añadir verificaciones durante las distintas fases que conllevan implícitamente el alta, baja y edición de los datos de las entidades.
- **Jackson Datatype:** Aporta al proyecto herramientas para sintetizar datos en formato JSON. Permite traducir objetos nativos java a formato JSON y viceversa.
- **DevTools:** Incluye al proyecto un conjunto de herramientas para facilitar en algunos aspectos la tarea de desarrollo.
- **HSQLDB:** Incluye una base de datos simple y embebida en el proyecto de manera provisional únicamente en el entorno de desarrollo.
- **PostgreSQL:** Incorpora los controladores necesarios para conectarse a una base de datos externa bajo el sistema gestor PostgreSQL.
- **Lombok:** Esta dependencia genera automáticamente los métodos genéricos que requiere un objeto de cualquier clase, como por ejemplo los *setters*, *getters* y `toString()`. De esta manera, el desarrollador no ha de invertir tiempo en ello además de ofrecer un código más limpio ya que de lo contrario, en la mayoría de los casos, se han de definir los dos primeros métodos nombrados para cada uno de los atributos de un objeto.
- **Apache HttpClient:** Permite al programa desarrollado bajo el *framework* actuar como un cliente y realizar peticiones HTTP a otros servidores.

4.3.2. Estructura

En esta sección se detallará la estructura y el contenido de los directorios más relevantes que forman el proyecto en cuestión partiendo des del directorio raíz. Para cada uno de ellos se dedicará una sección exponiendo la naturaleza de su contenido.

```
Configuration Complexity Abstraction
```



4.3.2.1. Directorio /config

Este directorio está destinado a albergar los ficheros relativos a la configuración general de la aplicación. En concreto los ficheros:

- **AuthenticationConfig.java:** Se encarga de generar un usuario y contraseña de acceso genérico a la aplicación.
- **CORSFilter.java:** Define los atributos permitidos en las cabeceras de las peticiones HTTP. En concreto especifica:
  - Los clientes permitidos para acceder a la API en base a su dirección IP: Se permite a cualquiera.
  - Los métodos permitidos: GET, POST, PUT, PATCH, DELETE, OPTIONS.
  - El tipo de parámetros incluidos en las cabeceras: *authorization* que permite añadir la autenticación en la cabecera y *content-type* que permite definir el formato del contenido de la respuesta.
- **WebSecurityConfig.java:** Permite definir si un usuario debe estar autenticado en base al método utilizado en la petición y al directorio de la API al que se accede. En este caso se requiere autenticación para la utilización de cualquier método HTTP en cualquier directorio a excepción del método OPTIONS que se permite sin autenticación y para cualquier directorio.

4.3.2.2. Directorio /controller

Este directorio contiene los ficheros con el código fuente que define para la aplicación las URI con funcionalidad alternativa a las generadas automáticamente por parte del framework Spring. La funcionalidad de las generadas automáticamente están destinadas al mantenimiento de los

objetos de las entidades definidas en el proyecto. Por lo tanto, para agregar funcionalidad extra a la API, como es el caso, se deben definir estas URI alternativas para, por ejemplo, obtener y enviar datos a los equipos objetivo del proyecto.

A continuación se detalla el cometido del código fuente de los ficheros contenidos en este directorio.

- **ApplyConfigurationController.java:** Se encarga de transmitir los datos con la nueva configuración recibidas de un cliente cualquiera a los equipos.
- **GetConfigurationController.java:** Se encarga de obtener los valores configurados y configurables de los equipos y responder con ello al cliente que lo haya solicitado.
- **SaveConfigurationController.java:** Se encarga de transmitir a los equipos la orden de respaldar la configuración en un directorio en concreto y, en caso de que el cliente lo solicite, de certificarla.

4.3.2.3. Directorio /domain

Contiene la definición del modelo de datos de las entidades a gestionar en la plataforma en formato de clases Java. La dependencia JPA mencionada anteriormente se encargará de sintetizar estos objetos además de las relaciones entre ellos y de definir las tablas y reglas correspondientes en la base de datos automáticamente. Los ficheros contenidos en este directorio corresponden a los siguientes.

- Building.java
- Campus.java
- Card.java
- Connector.java
- EquipmentRoom.java
- Equipment.java
- Floor.java
- Port.java

Como ya se ha comentado, corresponden a definiciones de clases Java básicas aunque añadiendo las etiquetas: `@OneToMany`, `@ManyToOne` y `@OneToOne` se definen las relaciones entre ellos.

Un ejemplo de ello es el siguiente fragmento de código fuente en el cuál, para definir la relación entre Campus y Building donde un Campus está relacionado con varios Buildings, la etiqueta `@OneToMany` precede a la lista que contiene estos últimos para establecer el tipo de relación que mantienen.

```
public class Campus extends UriEntity<Long> {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotBlank
    private String title;

    private String description;

    @OneToMany(mappedBy = "isInCampus", fetch = FetchType.EAGER)
    @Fetch(value = FetchMode.SUBSELECT)
    @JsonIdentityReference(alwaysAsId = true)
    private List<Building> buildings = new ArrayList<>();

}
```

4.3.2.4. Directorio /handler

Este directorio está destinado a albergar los ficheros de código fuente mediante los cuales se puede alterar las diferentes etapas que conlleva el alta, baja o modificación de un registro de cualquier entidad en la API.

En este caso sólo se ha requerido de uno. Dicho fichero corresponde a `CardEventHandler.java`, el cual captura el evento disparado justo cuando se ha creado un objeto tipo `Card` en la base de datos. En este momento genera tantos objetos de tipo `Port` como se haya especificado en el campo `numberOfPorts` y se relacionan con la `Card` que los alberga. En el momento que se da de baja una `Card` se ejecuta el mismo proceso pero inverso para eliminar los `Port` generados anteriormente.

Un ejemplo de ello es el siguiente fragmento de código del fichero `CardEventHandler.java` donde:

- `@RepositoryEventHandler`: Especifica sobre que entidad del modelo de datos ha de capturar los eventos, en este caso los relacionados con `Card`.
- `@HandleAfterCreate`: Especifica que evento ha de capturar, en este caso captura el evento disparado una vez se ha creado un objeto del tipo `Card`.
- `@HandleBeforeDelete`: Captura el evento diparado en el momento que se ha eliminado un objeto de tipo `Card`.

```
@RepositoryEventHandler(Card.class)
public class CardEventHandler {
    private final Logger logger =
            LoggerFactory.getLogger(CardEventHandler.class);

    @Autowired
    private PortRepository portRepository;

    @HandleAfterCreate
    public void handleCardAfterSave(Card card) {
        for(int i=1; i<=card.getNumberOfPorts(); i++){
            Port p = new Port();
            p.setTitle(String.valueOf(i));
            p.setPortNumber(i);
            p.setIsInCard(card);
            card.addPort(p);
        }
    }

    @HandleBeforeDelete
    public void handleCardBeforeDelete(Card card) {
        for(Port p: card.getPorts()){
            Connector c = p.getConnector();
            if(c != null){
                c.setConnectedTo(null);
            }
            p.setConnector(null);
```

```
        }
    }
```

4.3.3. Directorio /repository

Este directorio contiene la cabecera de las funciones de búsqueda de registros de las entidades a partir de algún atributo de estas. Spring es capaz de sintetizar el nombre de la función y mediante este auto-genera el código necesario para satisfacer el cometido de esta.

Se dispone de un fichero de este tipo para cada una de las entidades del modelo listadas anteriormente. Un ejemplo del contenido de un fichero de este tipo, en concreto, `BuildingRepository.java` corresponde al siguiente:

```
@RepositoryRestResource
public interface BuildingRepository
        extends PagingAndSortingRepository<Building, Long> {

    List<Building>
    findByTitleContainingIgnoreCase(@Param("title") String title);

    List<Building>
    findByTitleContainingIgnoreCaseAndIsInCampus(@Param("title")
            String title, @Param("campus") Campus campus);
}
```

En cuanto a cada una de las cabeceras Spring es capaz de interpretar y autogenerar el código necesario para:

- `findByTitleContainingIgnoreCase()`: Analizar el modelo de datos y devolver una lista de elementos de la clase `Building` cuyo título coincida con el del parámetro `title`.
- `findByDescriptionContaining()`: Analizar el modelo de datos y devolver una lista de elementos de la clase `Building` cuyo título coincida con el del parámetro `title` y además estén relacionados con el `Campus` especificado mediante el parámetro `campus`.

4.3.4. Directorio /utils

Este directorio contiene la definición de las clases que contendrán los valores configurados y configurables obtenidos del equipo para, pos-

teriormente traducirlos a formato JSON y responder a la petición del cliente. De esta manera es el propio Spring el que se encarga de mapear los atributos de las clases con los del texto en formato JSON de la respuesta.

4.3.5. Comunicación con el equipamiento

Una vez detallada la estructura del proyecto, se hará hincapié en una de las funcionalidades de este componente, el *back-end*, más importante dejando de lado la gestión de las entidades mediante la API REST generada. Esta funcionalidad corresponde a la de la comunicación con el equipamiento, y es que existe una línea muy fina que separa la implementación de este último con la de este propio componente. Así pues, en este apartado se expondrá como se maneja dicha comunicación por parte del *back-end*.

Tal y como se ha comentado, los ficheros que albergan esta funcionalidad se encuentran en el directorio `/controller` y, para cada uno de ellos se detallará, mediante un diagrama de secuencia su funcionalidad.

4.3.5.1. GetConfigurationController.java

El código fuente de este fichero se divide en dos funciones:

- `getAvailableSettings(connectorID)`
- `getCurrentSettings(connectorID)`

Se podrían integrar en una misma aunque de esta forma se pueden ejecutar de manera concurrente y agilizar la obtención de datos del equipo, reduciendo aproximadamente a la mitad el tiempo de ejecución. También cabe añadir que cada una de las funciones lanza varias peticiones aunque de manera encadenada, ya que de no ser así se superaría el límite establecido por el equipo de 4 sesiones simultáneas. Cabe añadir que se ha desestimado el paso de des-autenticación introducido en la etapa de diseño de la Comunicación con el equipamiento, ya que el propio equipamiento se encarga de eliminar las sesiones antiguas y con ello se ve decrementado el numero de peticiones realizadas.

El diagrama de secuencia de la Figura 4.2 refleja el cometido que desempeña la función `getAvailableSettings()` la cual, como ya se ha comentado, lanza peticiones con comandos CLI al equipo para conocer

los valores configurables.

En lo referente a *getCurrentSettings()* cabe mencionar que coincide con *getAvailableSettings()* pero ejecutando distintas instrucciones CLI en el equipo, en este caso para obtener la configuración actual.

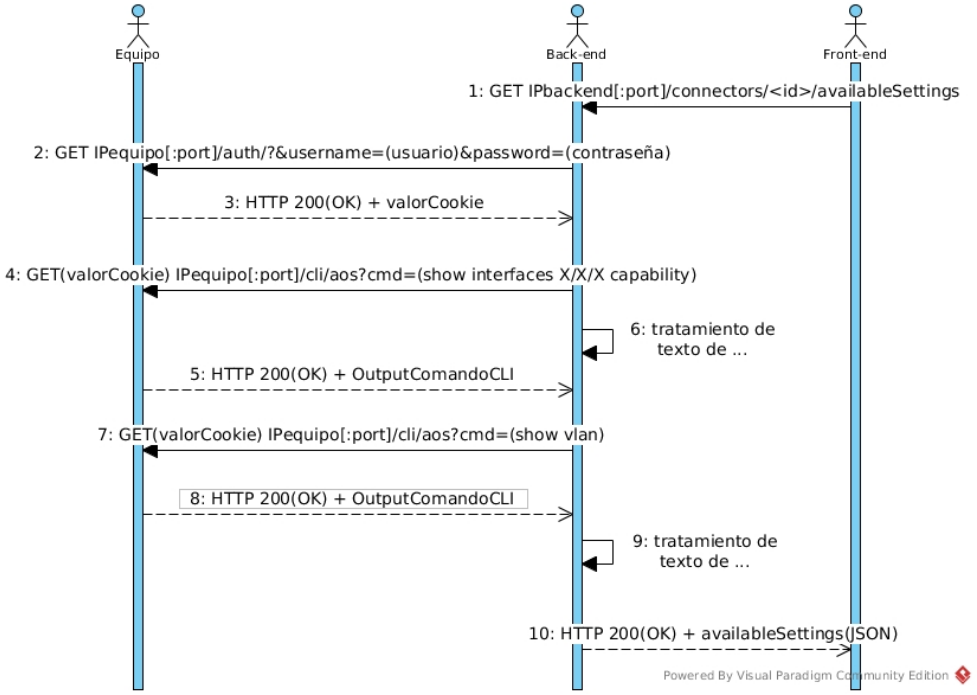


Figura 4.2: Diagrama de secuencia de la función `getAvailableSettings()`

4.3.5.2. ApplyConfigurationController.java

Este fichero contiene el código fuente con el que, a partir de los valores configurables obtenidos por el de la sección anterior, se aplican los cambios en la configuración del equipo.

Teniendo en cuenta que la plataforma está orientada a usuarios básicos se permite configurar los siguientes parámetros:

- Asignar un puerto a una VLAN ya creada.

- Cambiar el estado administrativo de un puerto a: encendido o apagado.
- Asignar un modo de Duplex a un puerto.
- Establecer la velocidad de un puerto.

Para cada una de las opciones de configuración listadas se destina un método perteneciente a la clase `ApplyConfigurationController.java`. Como en el fichero de la sección anterior, el esqueleto coincide en todos y cada uno de estos métodos aunque ejecutan comandos CLI distintos en los dispositivos. Por este motivo únicamente se aportará un diagrama de secuencia, el cuál se adjunta en la Figura 4.3 y pertenece al método `setVLAN()`.

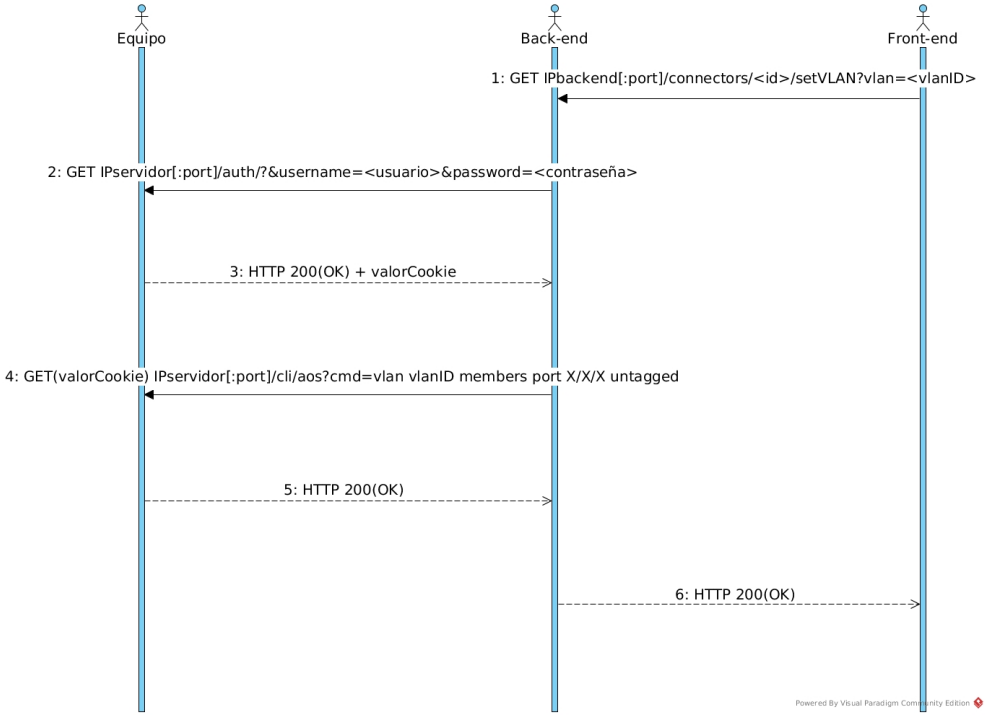


Figura 4.3: Diagrama de secuencia de la función `setVLAN()`

4.3.5.3. SaveConfigurationController.java

Una vez aplicados los cambios mediante las funciones del fichero de código fuente anterior los cambios resultan volátiles en el equipo. Esto

implica que en el próximo reinicio del equipo no se mantendrán. Para mantenerlos se puede almacenar la configuración actual en un directorio de respaldo para ser utilizada en otra ocasión. Además también se puede certificar dicha configuración para que sea con la que arranque el equipo tras el próximo reinicio.

Para llevar a cabo el respaldo y certificación de la documentación disponemos del método `saveAndOrCertify()` que recibe como parámetros: el id del conector, el directorio de respaldo y un booleano que especifica si hay que certificar o no la configuración.

El funcionamiento de este método se detalla mediante el diagrama de secuencia adjunto en la Figura 4.4.

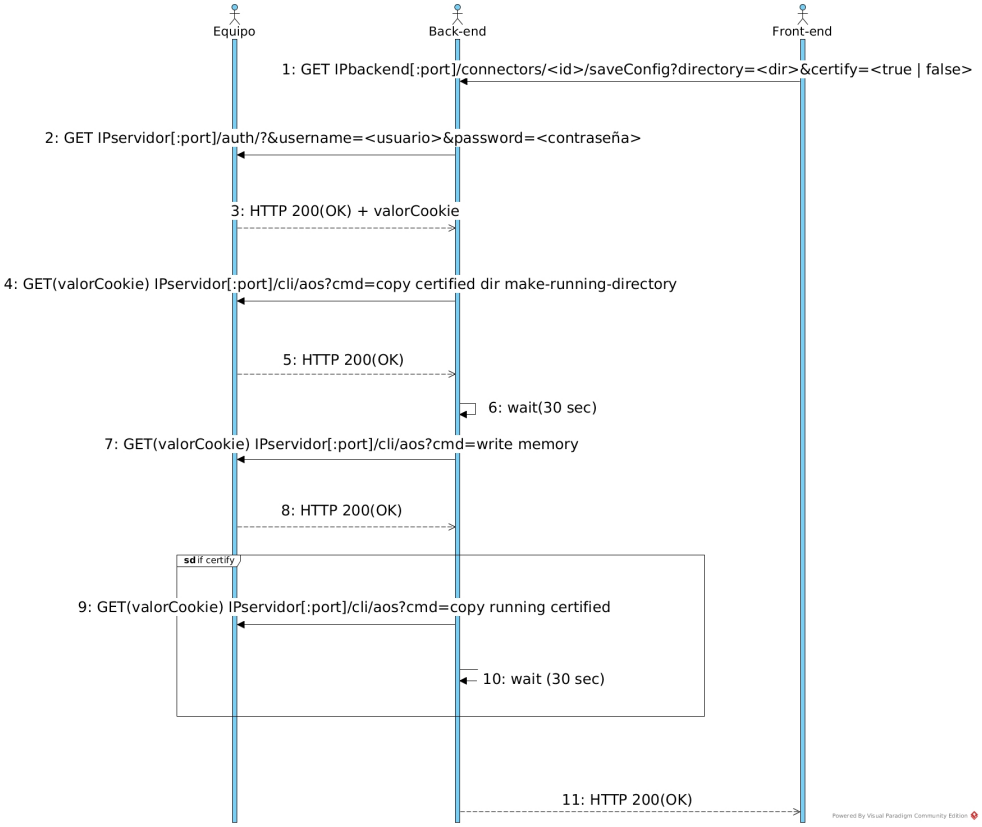


Figura 4.4: Diagrama de secuencia de la función `saveAndOrCertify()`

Cabe destacar las esperas establecidas tras lanzar los comandos CLI `copy certified <dir>make-running-directory` y `copy running`

`certified` en los puntos 6 y 10 del diagrama 4.4. Se deben a que los equipos responden a la petición en el momento que se lanza el comando tal y como se muestra en la Figura 4.5, en vez de cuando haya finalizado. Debido a ello se pueden generar inconsistencias en la configuración del equipo y, por ello se ha estimado el tiempo que se requiere lanzando estos mismos comandos a través del terminal del equipo,un ejemplo de ello queda reflejado en la Figura 4.6.

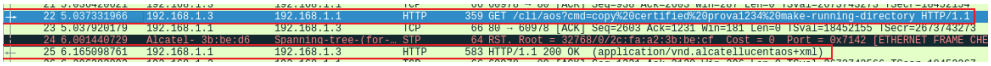


Figura 4.5: Ejecución a través de HTTP

```
-> copy certified provaSave make-running-directory
Please wait..
Fri Jan  3 03:26:51 : flashManager FlashMgr Main info message:
+++ Image file Uos.img differs - copying file
..............
Fri Jan  3 03:27:18 : ChassisSupervisor MipMgr info message:
+++ Copy certified succeeded


-> write memory

File /flash/provaSave/vcsetup.cfg replaced.

File /flash/provaSave/vcboot.cfg replaced.

-> copy running certified

Fri Jan  3 03:28:00 : flashManager FlashMgr Main info message:
+++ Verifying image directory provaSave on CMM flash
Please wait.............
Fri Jan  3 03:28:24 : ChassisSupervisor MipMgr info message:
+++ Copy running to certified succeeded


->
```

Figura 4.6: Ejecución a través de CLI

Podemos ver como en la Figura 4.5 el equipo responde a la petición en aproximadamente 1 segundo, mientras que en la captura 4.6 finaliza en alrededor de 27 segundos. Lo mismo sucede con el comando `copy running certified`

4.4. *Front-end*

En esta sección se da conclusión a la etapa de implementación. En ella se desarrollará el componente *front-end* el cual conjuntamente con el *back-end* y la comunicación con el equipamiento este proyecto devendrá funcional bajo el entorno de desarrollo.

4.4.1. Inicialización

Para llevar a cabo el desarrollo de un proyecto bajo Angular la primera tarea a realizar consiste en la inicialización del proyecto. Para ello se requiere tener ciertos paquetes instalados en el sistema ya que, tal y como se comentó en la correspondiente fase de análisis, este *framework* trabaja bajo el entorno Node.js.

El procedimiento para la instalación es el siguiente:

1. Instalar el paquete npm(*Node Package Manager*) ya que Angular trabaja bajo el entorno Node. Dicho paquete se consigue mediante el comando:

   ```
   sudo apt-get install npm
   ```

2. Tras instalar npm se aconseja actualizarlo a la última versión disponible mediante el comando:

   ```
   npm install npm@latest -g
   ```

3. Una vez descargado y actualizado el paquete npm se debe instalar el entorno Angular en este mediante el comando:

   ```
   npm install -g @angular/cli@latest
   ```

En este momento ya disponemos de todos los entornos exigidos, por lo que nos desplazaremos al directorio donde se desee inicializar el proyecto y se ejecutará el comando:

```
ng new <nombre del proyecto>
```

El resultado es un proyecto por defecto en Angular en el cual se incluyen los ficheros de configuración y librerías básicas de este *framework*.

4.4.2. Estructura

La estructura de directorios más relevantes des de la raíz que ofrece el proyecto corresponde a la siguiente:

```
Configuration Complexity Abstraction Client
├── /src
│   ├── /app
│   ├── /assets
│   └── /environments
└── package.json
```

4.4.2.1. Directorio /app

Contiene los ficheros de código fuente desarrollados y organizados en subdirectorios nombrados según la entidad a la que hace referencia coincidiendo con estas últimas definidas en Spring: `Building`, `Campus`, `Card`, `Connector`, `EquipmentRoom`, `Equipment`, `Floor`, `Port`.

El árbol de subdirectorios del directorio `/app` correspondiente a `/campus`, el cual coincide con el resto, es el siguiente:

```
/app
├── /building
├── /campus
│   ├── /campus-details
│   ├── /campus-edit
│   ├── /campus-form
│   ├── /campus-list
│   ├── /campus-search
│   ├── campus.service.ts
│   ├── campus.ts
│   └── update.campus.service.ts
├── /card
├── /connector
├── /equipmentRoom
├── /equipment
├── /floor
└── /port
```

4.4.2.2. Directorio /app/campus

En la fase de análisis ya se introdujo que Angular funcionaba mediante módulos y así es como se ha gestionado la estructura. Cada uno de los subdirectorios de `/app/campus` corresponden a un módulo de manera

que se pueden incrustar en otros.

Cada uno de los módulos dispone de un fichero `.ts`, `.html` y `.css`. El primero de ellos define la lógica de la vista, el segundo corresponde al esqueleto y el tercero al estilo. Esta misma estructura se repite en todos los módulos.

El cometido de cada uno de ellos queda detallado en la siguiente lista:

- **Campus Details:** Obtiene y muestra la información relativa a un objeto concreto de la entidad Campus.
- **Campus Edit:** Obtiene un objeto de la entidad Campus y se encarga de mostrar la información sobre este permitiendo ser editada, en este ultimo caso utiliza los mecanismos necesarios para actualizar el modelo de datos.
- **Campus Form:** Dispone del formulario de creación de un objeto de tipo Campus.
- **Campus List:** Obtiene todos los objetos Campus y los lista.
- **Campus Search:** Permite buscar y filtrar a partir del nombre y actualizar la Campus List.

También en el directorio `/campus` existen tres ficheros de código fuente y su cometido corresponde al siguiente:

- `campus.service.ts`: Se encarga de definir los mecanismos necesarios para la obtención, alta, baja y edición de los objetos Campus en la fuente de datos.
- `campus.ts`: Define los atributos del objeto Campus, de manera que estos se manejan empaquetados en un objeto y resulta mucho más cómodo el envío y recepción de objetos de este tipo mediante el formato JSON.
- `update.campus.service.ts`: Permite la comunicación entre módulos en tiempo de ejecución.

4.4.2.3. Directorio /assets

Este directorio contiene los ficheros multimedia utilizados en las vistas y el fichero `environment.json` en el que se establece la URL de acceso al `back-end`.

4.4.2.4. Directorio /environments

Contiene los ficheros que permiten parametrizar la ejecución de este componente según se ejecuten en el entorno de desarrollo o de producción. En este caso los utilizamos para especificar la URL mediante la cual se accede al *back-end* o API. Por ejemplo el fichero llamado `environment.prod.ts` ofrece el siguiente contenido:

```
  export const environment = {
  production: true,
  API: 'http://localhost:8080'
};
```

4.4.2.5. Fichero package.json

Este fichero contiene la definición de las dependencias utilizadas durante el desarrollo del proyecto, por lo que también serán necesarias para su ejecución. Dejando de lado las que incorpora el proyecto por defecto, se han añadido las dependencias adicionales listadas y detalladas a continuación:

- `Bootstrap`: Permite estructurar en forma de tablas los elementos de las vistas, además de aportar mecanismos para redistribuir automáticamente dichos elementos en pantallas de diferentes tamaños.
- `ng2-img-map`: Permite establecer zonas reactivas en una imagen determinada mediante coordenadas.
- `ng2-toastr`: Permite mostrar mensajes deslizantes en la interfaz de usuario.

4.4.3. Conexión con el back-end

Tal y como se ha comentado en otros apartados, este componente ofrece una interfaz visual para el manejo de datos del *back-end* además de la configuración con el equipamiento. Cabe remarcar que esta última

se realiza también a través del *back-end* como intermediario. En esta sección se detalla como se lleva a cabo esta tarea.

Cabe añadir que la autenticación entre los dos componentes se realiza a través del protocolo `O2Auth`. Consiste en que el *back-end* genera un *token* a partir de las credenciales de autenticación, lo transmite al *front-end* y este ultimo lo incluirá en las futuras peticiones.

El alta, baja y edición del modelo de datos se realiza mediante los ya nombrados módulos Details, Edit, Form, List y Search. No obstante para manejar la configuración de los equipos se realiza a través del código fuente desarrollado en `/app/connector-list/modal-config` el cual puede realizar las siguientes operaciones:

- Obtener valores configurados.
- Obtener valores configurables.
- Establecer el estado administrativo del puerto.
- Asignar el puerto a una VLAN diferente.
- Establecer la velocidad de conexión.
- Establecer el modo del Duplex.
- Respaldar y certificar los cambios de la configuración en un directorio.

La lógica de estas operaciones se encuentra en el fichero `modal-config.ts` cuyo comportamiento se resume mediante el diagrama de secuencia de la Figura 4.7.

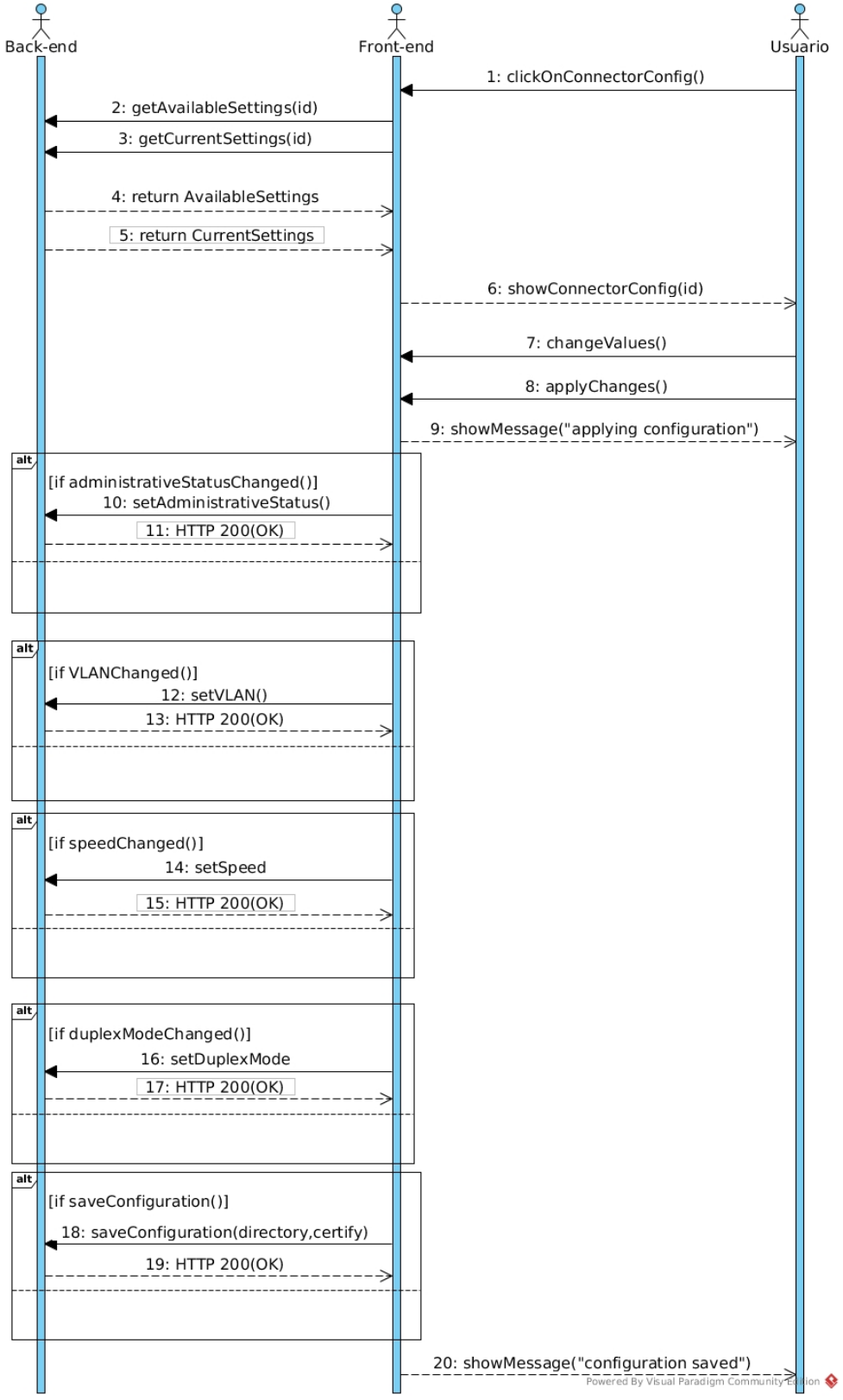


Figura 4.7: Diagrama de secuencia para actualizar la configuración

4.5. Interfaz de usuario

Dejando de lado los formularios para inicializar los datos relativos al patrimonio arquitectónico y equipamiento tecnológico para inicializar la plataforma, se hará hincapié a la principal funcionalidad de la aplicación: la selección de puntos de red en el plano y la pantalla de configuración de cada uno de ellos; para ello se recurre a las capturas de pantalla reflejadas en las figuras 4.8 y 4.9 respectivamente. En la primera de ellas se puede observar el listado de puntos de red de una planta, además dichos puntos se ven reflejados en el plano. En la segunda se muestra el menú de configuración tras haber seleccionado un punto de red cualquiera del conjunto de la pantalla anterior.

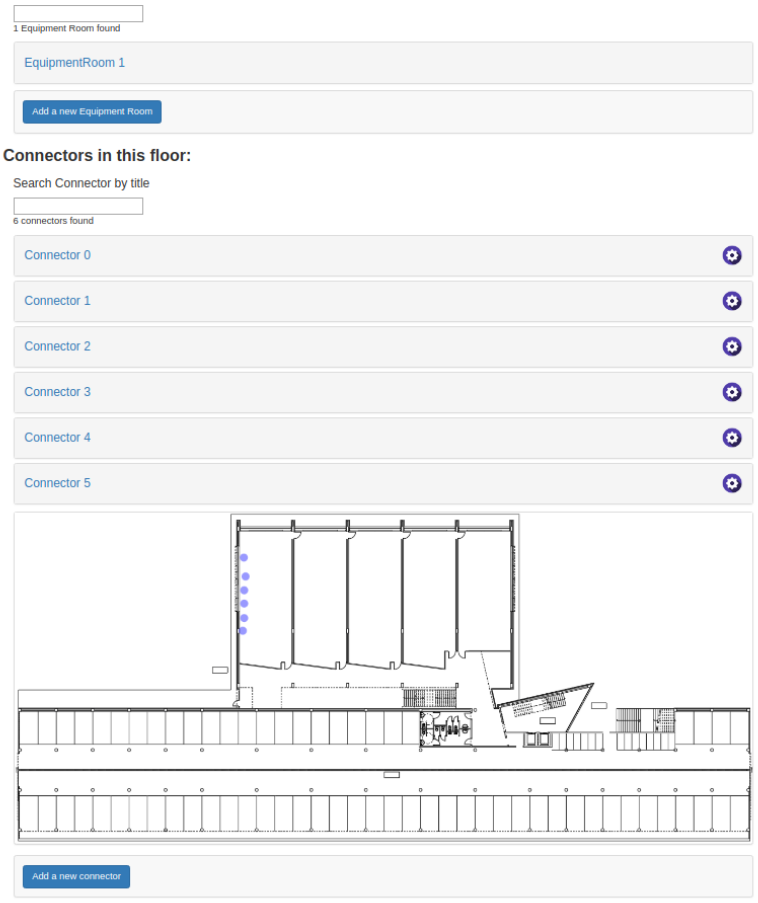


Figura 4.8: Interfaz: Listado y plano de los puntos de red

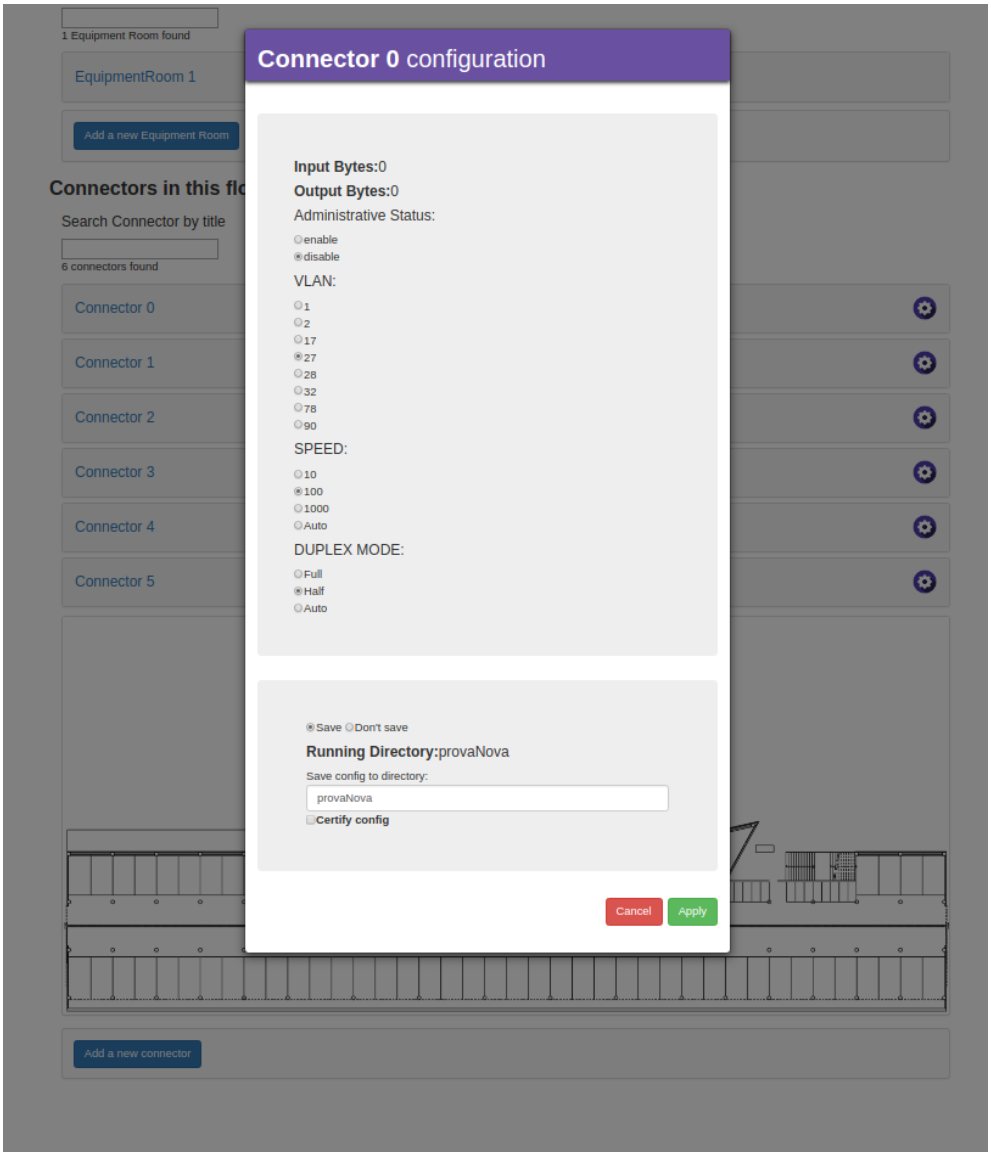


Figura 4.9: Interfaz: Menú de configuración de un punto de red

Capítulo 5

Implantación

La implantación constituye la fase final del proyecto, en la cuál se despliegan en el servidor que los albergará los componentes resultantes de la fase anterior correspondiente a la de implantación.

A continuación se va a detallar el procedimiento a seguir para implantar cada uno de los componentes en un equipo con el rol de servidor. Cabe añadir que la arquitectura de la plataforma es muy flexible, de manera que se permite implantar todos los componentes en un mismo servidor o bien en varios(tantos como componentes). El único requisito es que puedan establecer y mantener una comunicación de red con sus respectivos componentes adyacentes.

El laboratorio de trabajo utilizado para llevar a cabo el despliegue de los componentes corresponde a un equipo con las siguientes características:

- Sistema operativo: Ubuntu 17.04
- Version de java: 1.8.0_131

5.1. *Back-end*

En esta sección se detallará el procedimiento a seguir para la implantación del *back-end*. Se considera un requisito que el servidor que albergará este componente esté conectado en un puerto el cual tenga asignada la VLAN de gestión del equipamiento.

5.1.1. Base de datos

Cómo el *back-end* depende directamente de una base de datos PostgreSQL el primer paso será crearla e inicializarla. Este procedimiento resulta imprescindible ya que de lo contrario el ejecutable no puede ni tan siquiera ponerse en marcha. Para ello hay que realizar las siguientes operaciones en el terminal del servidor:

1. Instalar el paquete que maneja el entorno PostgreSQL mediante el siguiente comando:

   ```
   sudo apt-get install postgresql postgresql-contrib
   ```

2. Inicializar la autenticación al entorno PostgreSQL. Para ello se lanzará el siguiente comando que establecerá la contraseña de acceso del usuario por defecto para acceder a dicho entorno:

   ```
   sudo -u postgres psql -c "ALTER USER postgres PASSWORD
                        'password';"
   ```

3. Debido a que el entorno crea un usuario nuevo en el sistema operativo, hay que establecer también una contraseña para este mediante el siguiente comando:

   ```
   sudo passwd postgres
   ```

4. Para terminar se crea una nueva base de datos en el entorno PostgreSQL destinada exclusivamente para el *back-end* del proyecto mediante el siguiente comando:

   ```
   createdb -h localhost -p 5432 -U postgres
   ConfigurationComplexityAbstraction password
   ```

   Este comando crea una base de datos con los siguientes parámetros:

   - `-h` se especifica en que servidor se creará la base de datos, ya que se puede lanzar en un servidor remoto. En este caso `localhost` corresponde al propio equipo en el que se lanza el comando.
   - `-p` se especifica a través de qué puerto será accesible la base de datos.
   - `-U` corresponde al usuario con el que se autenticará en el entorno PostgreSQL para crear dicha base de datos, en este caso `postgres`.
   - El nombre de la base de datos destinada al proyecto corresponde a `ConfigurationComplexityAbstraction`.

- El último parámetro corresponde a la contraseña utilizada junto al parámetro `-U` para la autenticación.

5.1.2. Establecer el *back-end* como un servicio

Una vez inicializada la base de datos podemos lanzar el ejecutable del *back-end* disponible desde el link `https://github.com/jmelich/SwitchConfigurationBackendReleases/releases/download/final/SwitchConfiguration.jar`. El comando básico para ponerlo en marcha coincide con el de cualquier fichero de este tipo:

```
./SwitchConfiguration.jar.
```

Las aplicaciones web requieren normalmente de una aplicación que ponga en marcha los servicios web y manejen las peticiones, por ejemplo: Apache Tomcat. En el caso de Spring no es necesario ya que este lleva un servidor de este tipo embebido que se pone en marcha al lanzar el ejecutable.

Además, suele interesar que las aplicaciones de esta naturaleza se ejecuten automáticamente al arrancar el servidor, por lo tanto devienen un servicio. Para ello, en este caso utilizaremos `systemd`, que se encarga de gestionar los servicios que se ejecutan al arrancar el sistema y el procedimiento es el siguiente:

1. Situarse al directorio que albergará los ficheros de la aplicación. Por ejemplo:

   ```
   cd /var/SwitchConfiguration
   ```

2. Una vez dentro del directorio descargar el fichero ejecutable `.jar` mediante el comando:

   ```
   wget https://github.com/jmelich/
   SwitchConfigurationBackendReleases/releases/
   download/final/SwitchConfiguration.jar
   ```

   En caso de que no se disponga de conexión a internet se puede obtener del directorio adjunto a la documentación, en la ruta:

   ```
   SwitchConfigurationProject/back-end/bin/
   SwitchConfiguration.jar
   ```

3. Establecer permisos de ejecución para el fichero ejecutable:

```
chmod +x SwitchConfiguration.jar
```

4. Crear un fichero de configuración **opcional** con el nombre SwitchConfiguration.conf, de esta manera `systemd` comprueba automáticamente si existe un fichero de configuración para el ejecutable. Permite varios parámetros de configuración detallados en [10], aunque se va a hacer hincapié al parámetro `RUN_ARGS` que nos permitirá parametrizar el *back-end*. En este caso el fichero contendría una única línea cómo la siguiente:

   ```
   RUN_ARGS=[parametro0] [parametro1] ... [parametroN]
   ```

   de manera que se encadenan los parámetros separados por un espacio. Dichos parámetros pueden ser los siguientes:

   - `-spring.datasource.url=<urlDB>` especifica la localización de la base de datos. `urlDB` ha de tener el siguiente formato: `jdbc:postgresql://<ip>:<puerto>/<nombreDB>`. El valor por defecto es: `jdbc:postgresql://localhost:5432/configurationComplexityAbstraction`
   - `-spring.datasource.username=<usuario>` especifica el usuario de acceso a la base de datos. El valor por defecto es: `postgres`
   - `-spring.datasource.password=<contraseña>` especifica la contraseña de acceso a la base de datos. El valor por defecto es: `password`
   - `-server.port=<puerto>` permite modificar el puerto del *back-end* que maneja las peticiones. El valor por defecto es: `8080`

5. Crear un fichero de servicio llamado `SwitchConfiguration.service` en el directorio `/etc/systemd/system` e insertar las siguientes líneas en él:

   ```
   [Unit]
   Description=SwitchConfiguration
   After=syslog.target

   [Service]
   ExecStart=/var/SwitchConfiguration/SwitchConfiguration.jar
   SuccessExitStatus=143
   ```

```
[Install]
WantedBy=multi-user.target
```

6. Añadir este servicio a la lista de los que ejecuta `systemd` tras el arranque lanzando el comando:

```
systemctl enable SwitchConfiguration.service
```

5.1.3. Seguridad

Una vez configurado el *back-end* cómo un servicio, de manera opcional aunque recomendable se puede reducir la vulnerabilidad de los ficheros ante ataques informáticos mediante el siguiente procedimiento:

1. Se cambiará el propietario del fichero, ya que este nunca debe ser el superusuario lanzando el comando:

```
chown <usuario>:<usuario>/var/SwitchConfiguration/
SwitchConfiguration.jar
```

2. Se garantizará los permisos de lectura y ejecución únicamente al propietario del fichero mediante el comando:

```
chmod 500
/var/SwitchConfiguration/SwitchConfiguration.jar
```

3. Se establecerá el fichero de código fuente como inmutable para evitar que sea modificado de forma fraudulenta mediante el comando:

```
chattr +i
/var/SwitchConfiguration/SwitchConfiguration.jar
```

5.2. *Frontend*

En esta sección se detallará el procedimiento a seguir para la implantación del *front-end* con lo cual la plataforma web resultante del proyecto devendrá funcional. Se considera un requisito indispensable que el servidor que albergará este componente pueda establecer una conexión con el *back-end* a través de la red.

Una vez realizado el despliegue si no se ha alterado ninguno de los parámetros de configuración de por defecto se accederá a la plataforma a través de la URL:

`http://<URL_o_IPFrontend>:4200`

Usuario: `user`

Contraseña: `password`

5.2.1. Ejecución del *front-end*

En este caso, a diferencia del despliegue del *back-end* se requiere de un servidor HTTP que atienda las peticiones. Para ello utilizaremos uno de los que ofrece el propio entorno `npm` especializado en la ejecución de entornos bajo la arquitectura SPA.

Para ello deberemos instalar el entorno `npm` de la misma forma que se detalló en la fase de implementación de este mismo componente y, a posteriori, instalar el servidor HTTP mediante el siguiente procedimiento:

1. Instalar el paquete npm(*Node Package Manager*) mediante el siguiente comando:

   ```
   sudo apt-get install npm
   ```

2. Actualizarlo a la última versión disponible mediante el comando:

   ```
   npm install npm@latest -g
   ```

3. Instalar el servidor HTTP mediante el siguiente comando:

   ```
   npm install http-server-spa -g
   ```

Tras instalar el servidor HTTP podemos descargar el proyecto comprimido desde el link `https://github.com/jmelich/SwitchConfigurationFrontendReleases/releases/download/final/SwitchConfigurationClient.tar.gz`, extraer el contenido que corresponde a un directorio llamado `dist` y lanzar-lo desde el directorio donde se ha descomprimido mediante el comando:

`http-server-spa ./ / <puerto_a_utilizar>`

También, como en el caso del *back-end*, puede interesar que sea ejecutado como un servicio, por ello se detalla el procedimiento en la siguiente sección.

5.2.2. Establecer el *front-end* como un servicio

Para establecer el *front-end* como un servicio en el equipo se utilizará el mismo procedimiento que para el *back-end*, mediante `systemd`. En este caso también se requiere tener instalado el servidor HTTP de la sección anterior. En cuanto al procedimiento resulta muy parecido al del *back-end* aunque con pequeñas variaciones que se detallan a continuación:

1. Situarse en el directorio que albergará los ficheros de la aplicación. Por ejemplo:

   ```
   cd /var/SwitchConfigurationClient
   ```

2. Una vez dentro del directorio descargar el proyecto comprimido ya que consta de varios ficheros mediante el siguiente comando:

   ```
   wget https://github.com/jmelich/
   SwitchConfigurationFrontendReleases/releases/
   download/final/SwitchConfigurationClient.tar.gz
   ```

   En caso de que no se disponga de conexión a internet se puede obtener del directorio adjunto a la documentación, en la ruta:

   ```
   SwitchConfigurationProject/front-end/
   SwitchConfigurationClient.tar.gz
   ```

3. Descomprimir el contenido del fichero ejecutable con el que obtendremos el directorio `/dist`:

   ```
   tar -xzvf SwitchConfigurationClient.tar.gz
   ```

4. Especificar la URL a través de la cual se localiza el *back-end* en el fichero `/var/SwitchConfigurationClient/dist/assets/environment.json`.
   El valor por defecto corresponde a: `http://localhost:8080`.

5. Crear un fichero de servicio llamado `SwitchConfigurationClient.service` en el directorio `/etc/systemd/system` e insertar las siguientes líneas en él:

   ```
   [Unit]
   Description=SwitchConfigurationClient
   After=syslog.target

   [Service]
   Environment='PROJECT=../../../var/
   ```

```
SwitchConfigurationClient/dist'

Environment='PORT=4200'

ExecStart=/usr/bin/http-server-spa ${PROJECT}
index.html ${PORT}

SuccessExitStatus=143

[Install]
WantedBy=multi-user.target
```

- `PROJECT` corresponde a la ruta relativa desde este fichero hasta el proyecto a ejecutar.
- `PORT` corresponde al puerto de escucha del `front-end`.

6. Añadir este servicio a la lista de los que ejecuta `systemd` tras el arranque lanzando el comando:

   ```
   systemctl enable SwitchConfigurationClient.service
   ```

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

La realización de este trabajo me ha servido para profundizar y acabar de asentar los conocimientos previos en cuanto al desarrollo orientado a web y redes y comunicaciones, coincidiendo estas últimas con las dos vertientes tecnológicas que segregan el proyecto. En cuanto a la primera de ellas, concretamente Angular con el que ya había tenido una previa toma de contacto y me pareció imponer una curva de aprendizaje bastante pronunciada. Este hecho hizo que me replanteara si el proyecto en cuestión era el que debía tomar o quizá debía elegir otro. Aún así, no me di por vencido y seguí con él, con el objetivo de profundizar el bagaje de conocimientos en cuanto al desarrollo web y equilibrarlos con los que disponía en cuanto al desarrollo de aplicaciones de escritorio, que doy por logrado. Para la segunda vertiente correspondiente a las redes y comunicaciones hay que decir que, gracias al conocimiento adquirido durante la asignatura de redes, en la que se realizaban pruebas a nivel de laboratorio con equipos del mismo fabricante, se ha sabido manejar el equipamiento con una gran soltura. De lo contrario, la realización del proyecto hubiera resultado mucho más compleja. No obstante me he familiarizado aún más con el equipamiento y además he conocido un abanico de nuevas funcionalidades y configuraciones que estos últimos ofrecen.

En general, los principales retos con los que me he encontrado durante el desarrollo dejando de lado el aprendizaje han sido los siguientes:

- Determinar que el equipamiento devolvía en algunos casos, las respuestas a las peticiones antes de haber finalizado su ejecución. Por ello se generaban inconsistencias durante el respaldo de la configuración. Para dar con ello se estudió la relación entre los paquetes

transmitidos y la demora a obtener una respuesta mediante la herramienta Wireshark. Una vez detectado este comportamiento se estableció un tiempo de espera prudencial entre el lanzamiento de los comandos conflictivos.

- Manejar las entidades y relaciones del modelo de datos para tener en cuenta que los equipos, de manera virtual pueden representarse de manera única(*stacks*) y a su misma vez estar formados por diferentes tarjetas físicas. Todo ello se debe tener en cuenta para aplicar las configuraciones al puerto correspondiente.

- Decidir de que forma se lanzarían las peticiones al equipamiento, ya que las 4 sesiones simultáneas que ofrecen no son suficientes cómo para atender a todas las peticiones que se han de lanzar contra ellos des del *back-end* para obtener los datos relativos a la configuración. Por ello se utiliza una sesión para la obtención de la configuración actual y otra para obtener los valores configurables.

- Saber qué, de que manera y dónde buscar el modo de resolver los problemas o nuevas funcionalidades con los que me he ido viendo durante el desarrollo. Ya que al tratarse de tecnologías nuevas para mi, no daba con el concepto preciso a investigar.

- Durante los inicios fue un reto establecer una comunicación satisfactoria entre el *back-end* y el equipamiento debido a que la herramienta Postman de la que ya se ha hablado durante la fase de implementación, a menos que actives una opción poco visible da por sentado que en las peticiones de respuesta no se han incluido *cookies*. Por este motivo fallaban los posteriores intentos de conexión con el equipamiento con el motivo: *"Por favor, debe autenticarse primero"*.

- Durante el desarrollo de los módulos de Angular los cuales como ya se introdujo en la fase de Análisis pueden tener una granularidad tan fina como se desee, por ello fue un reto decidir para cada uno de ellos el grosor de funcionalidad que debía abarcar, ya que otorgar de más o de menos generaba inconsistencias al ser reutilizados en diferentes emplazamientos.

- Referente al componente mencionado en el punto anterior, supuso un reto escoger y adaptar los módulos desarrollados por terceros que se han integrado en la plataforma, ya que son genéricos y se deben moldear de manera que hasta el último momento no se percibe

si su funcionamiento podrá satisfacer la funcionalidad que se está desarrollando. Ejemplo de ello la representación de los puntos de red en un plano, los mensajes flotantes y la vista modal para representar la configuración de los puntos de red.

La resolución de los problemas detallados anteriormente entre otros ha permitido seguir con el desarrollo del proyecto hasta darlo por finalizado. Llegados a este punto se ha conseguido: un mayor dominio de las tecnologías en el ámbito web, una mejor gestión en cuanto a la metodología de desarrollo de un proyecto, un mayor dominio y nuevos conocimientos en cuanto al funcionamiento de las redes y comunicaciones, a entender y sintetizar la arquitectura y topología de una infraestructura de red corporativa y, a aprovechar el potencial de distintas tecnologías para obtener un único resultado.

6.2. Trabajo futuro

En cuanto al proyecto en cuestión cabe mencionar que cumple con los requisitos establecidos previamente: está desarrollado con software libre y permite la administración básica de equipos de acceso. Además el propio proyecto da lugar a nuevas ideas o mejoras ya que esta metodología se podría utilizar como plantilla para ampliar el abanico de opciones configurables y realizar la administración total de los equipos a través de esta plataforma. La seguridad es un factor importante en todos los ámbitos de la tecnología, es por ello que ambas de las propuestas que se realizarán en las siguientes secciones están relacionadas con ella. La primera de ellas corresponde a una nueva funcionalidad mientras que la segunda corresponde a una medida de seguridad para la plataforma.

6.2.1. Gestión de seguridad de puertos de acceso

Aunque no corresponda a un aspecto básico de configuración, en alguna ocasión se hace necesario limitar el acceso a un puerto.Por ello se propone habilitar la configuración relativa a LPS(*Learned Port Security*) que ofrece el equipamiento. Consiste en un mecanismo que permite especificar a qué dispositivos se les permitirá utilizar un determinado puerto en base a su dirección MAC(*Media Access Control*). Además se puede establecer un determinado comportamiento al puerto en el caso que se viole esta restricción. La funcionalidad que ofrece el mecanismo LPS se detalla a continuación:

- Permite establecer un contador de tiempo el cuál mientras esté activo, el puerto aprenderá las direcciones MAC utilizadas a través de este último y serán las que se permitirán en un futuro.
- Permite especificar el numero máximo de direcciones MAC que puede aprender un puerto.
- Como alternativa al primer punto se permite establecer las MAC permitidas en una lista.
- Permite definir el comportamiento del puerto en el caso de que se haya violado la restricción. En concreto:
  - Apagar el puerto.
  - Desactivar el aprendizaje al detectar una MAC no permitida o si se alcanza el límite establecido
  - Ignorar únicamente el tránsito de las MAC no permitidas.

En el caso de que se llevara a cabo la implementación de la configuración de LPS a través de la plataforma, a grandes rasgos, el procedimiento sería el siguiente:

1. Obtener la configuración actual de LPS en el equipamiento des del *back-end*.
2. Obtener los valores configurables de LPS en el equipamiento des del *back-end*.
3. Habilitar el *back-end* para recibir nuevas configuraciones de LPS des del *front-end*
4. Habilitar el *front-end* para la recepción de los valores de los dos primeros puntos.
5. Añadir una sección para LPS en el menú de configuración de los puntos de red en el *front-end*.
6. Habilitar el *front-end* para transmitir nuevas configuraciones de LPS al *back-end*.

6.2.2. Seguridad en la comunicación

Aunque a priori se considera que la red de gestión del equipamiento es segura se propone sustituir el protocolo HTTP utilizado actualmente para la comunicación entre los componentes(equipamiento-*back-end*-*front-end*) por HTTPS, la versión securizada de HTTP. Con ello aumentará el nivel de seguridad ya que en las comunicaciones realizadas bajo esta versión se cifra la mayor parte del contenido de las peticiones por parte del emisor. Véase la diferencia en las figuras 6.1 y 6.2 donde se muestra el contenido leíble o texto plano de una petición a través de HTTP y HTTPS respectivamente interceptadas mediante la herramienta Wireshark.

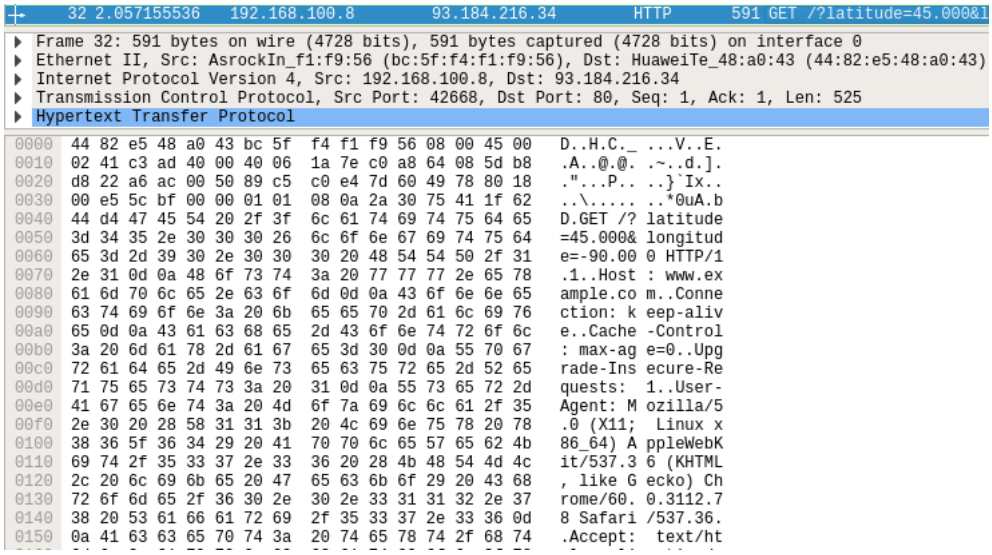


Figura 6.1: Petición HTTP

Según [4] este incremento en la seguridad se consigue transmitiendo las peticiones HTTP a través de un canal bajo el protocolo SSL(*Secure Sockets Layer*) o de su sucesor TLS(`Transport Layer Security`), del mismo modo que se transmite HTTP a través de TCP. Su principal característica consiste en que para establecer la comunicación, tanto el emisor como el receptor disponen de un certificado formado por una clave pública y una privada, de manera que realizando un intercambio de la clave pública y cifrando el contenido de las peticiones mediante esta última se asegura que únicamente el receptor podrá transformar el

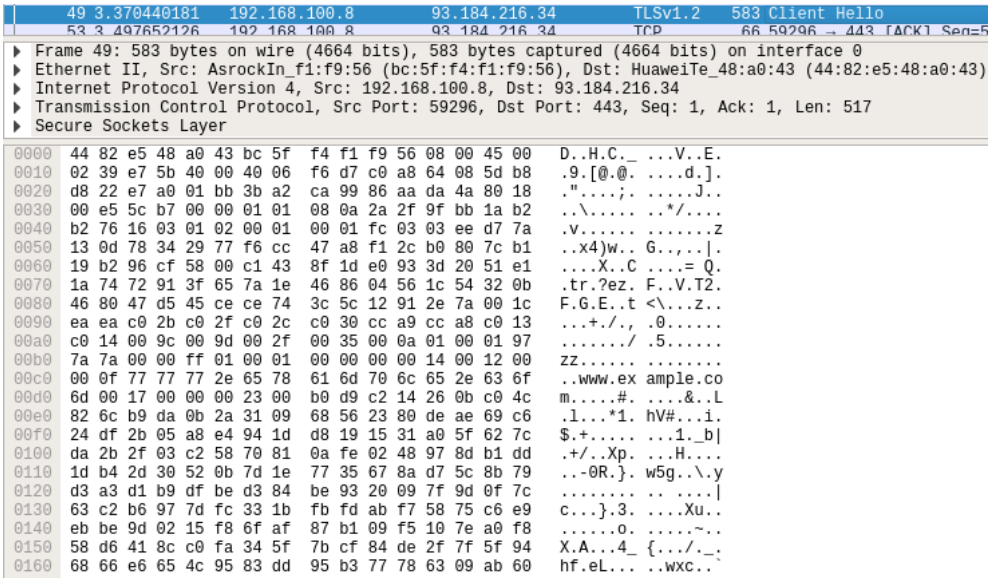


Figura 6.2: Petición HTTPS

contenido cifrado a texto plano.

Las modificaciones a realizar en el proyecto en cuestión para la utilización de HTTPS, a grandes rasgos consistirían en:

- **Equipamiento:**Activar SSL en los servicios web del equipamiento lanzando el comando CLI:

  ```
  webview force-ssl enable
  ```

  En este caso no se requiere generar u obtener un nuevo certificado ya que ya lo incorpora el equipamiento por defecto.

- ***Back-end:*** Como ya se comentó en la fase de diseño de este componente se ve involucrado en dos arquitecturas cliente-servidor, es por ello que debe configurarse para ambos sentidos.

  1. Generar un certificado que incluya el par de claves pública y privada y almacenar-lo en el directorio apropiado del proyecto para utilizarlo próximamente.
  2. (*back-end*<->equipamiento): Sustituir las librerías HTTP para realizar peticiones al equipamiento a través de librerías

HTTPS, especificando en estas últimas el certificado generado anteriormente. Además sustituir `http` por `https` a las plantillas URL.

3. (*back-end*<->*front-end*): Especificar en la configuración del servidor HTTP de Spring(Tomcat embebido) que recibirá las peticiones a través de HTTPS utilizando el certificado generado anteriormente.

- ***Front-end:*** Especificar el certificado a utilizar al servidor HTTP que atenderá las peticiones de este componente, en este proyecto corresponde http-server-spa. Para su conexión con el *back-end* únicamente se requiere sustituir `http` por `https` a las plantillas URL.

6.2.3. Miscelánea

Aparte de las mejoras relacionadas con la seguridad anteriores se proponen nuevas utilidades a implementar para facilitar las tareas de administración.

- Permitir la carga de ficheros relativos a Autocad, a los cuales se analizaría el contenido y se generarían los puntos de red y se darían de alta en la base de datos de la plataforma automáticamente.

- Desarrollar una herramienta para realizar copias de seguridad y restaurarlas.

- Permitir una gestión de usuarios de acceso a la plataforma. Incluso realizarla a través del servidor LDAP(*Lightweight Directory Access Protocol*) de la entidad.

Bibliografía

[1] Angular documentation. https://angular.io. Accessed: 2017-05-19.

[2] Ember documentation. https://guides.emberjs.com. Accessed: 2017-05-20.

[3] Google trends. https://trends.google.com/trends/explore?date=today%205-y,today%205-y,today%205-y,today%205-y&geo=,,,&q=ember.js,angular4,react.js,polymer.js&hl=en-US. Accessed: 2017-07-10.

[4] IETF http over tls. https://tools.ietf.org/html/rfc2818. Accessed: 2017-09-01.

[5] .NET documentation. https://docs.microsoft.com/en-us/dotnet/articles/framework/get-started/. Accessed: 2017-04-21.

[6] Nodejitsu single page application. https://blog.nodejitsu.com/single-page-apps-with-nodejs/. Accessed: 2017-04-21.

[7] PHP documentation. http://php.net/manual/en/history.php.php. Accessed: 2017-04-21.

[8] Polymer documentation. https://www.polymer-project.org/2.0/docs/devguide/feature-overview. Accessed: 2017-05-20.

[9] React documentation. https://facebook.github.io/react/docs. Accessed: 2017-05-20.

[10] Spring deployment. https://docs.spring.io/spring-boot/docs/current/reference/html/deployment-install.html. Accessed: 2017-10-23.

[11] Spring initializr. https://start.spring.io/.

[12] SpringData REST documentation. `http://projects.spring.io/spring-data-rest/#quick-start`. Accessed: 2017-04-21.

[13] TIOBE programming language popularity index. `https://www.tiobe.com/tiobe-index/`. Accessed: 2017-07-05.

[14] *Symfony, The book*. Apress, 2.8 edition, 2016.

[15] Jacob Kaplan-Moss Adrian Holovaty. *The Definitive Guide to Django*. Apress, 1 edition, 2008.

[16] Alcatel-Lucent. *OmniSwitch AOS Switch Management Guide*, 8 edition, 11 2015.

[17] Amuthan G. *Spring MVC Beginners Guide*. PACKT publishing, 1 edition, 2014.

[18] Amos Q. Haviv. *MEAN Web Development*. PACKT publishing, 1 edition, 2014.

[19] IEEE. *Software-Defined Networking: A Comprehensive Survey*, 1 2015.

[20] Mark Lutz. *Learning Python 5E*. O'reilly, 5 edition, 2013.

[21] Daniel Leuk Patrick Niemeyer. *Learning Java*. O'reilly, 4 edition, 2013.

[22] Roger S. Pressman. *Ingeniería del Software*. McGraw-Hill, 7 edition, 2010.

[23] M. Sendín. Enginyeria de programari, tema 1(part 2). 2016-2017.

[24] Cèsar Fernández y Enric Guitart. Xarxes i comunicacions, network management. 2015-2016.

[25] Marta Oliva y Juan Manuel Jimeno. Ampliació de bbdd i ep, nosql. 2015-2016.